NEC

PC-100 Series

N$_{100}$-BASIC ユーザーズガイド

PC-10000-BM1
PTS-257

ユーザーズガイド

PC-100 Series

N_{100}-BASIC

NEC

目　次

第5章　漢字処理 —— 69

第6章　データファイル入出力 —— 77

第7章　デバッグの基本 —— 89

第11章　チャイルドプロセスのサポート――123

付録――131

ご注意

1. ディスクファイルをOPENした場合は，END文，CLOSE文あるいはRESET文によってすべてのファイルを閉じるまで，ディスクをドライブから抜いたり，電源を切ったりしないようにしてください．このようなことをすると，ファイルが破壊されてしまいます．

2. プログラムの中で，同名のラベルを重複して使用すると，"Duplicate label"エラーが生じますが，いったんこのエラーが生じると，以後大文字でのコマンド入力はすべて同じエラーを起こしてしまいます．このようなときは，小文字の入力によって重複したラベルを削除するか，プログラム全体を削除(newコマンドによる)すればもとの状態に戻ります．

3. ディスプレイ画面に対してエスケープシーケンスを使用した場合，スクリーンエディタが正常に動作しなくなることがあります．このようなときは，SCREEN文，WIDTH文あるいはWIDTH USING文によって，現在の画面モード以外のモードを選択すればスクリーンエディタは正常状態に復帰します．ただし，これによっても正常にならないときには，BASICを再起動しなくてはなりません．

4. SHELLコマンドにより画面上に出力されたデータや文字に対してはスクリーンエディタによる編集を加えることはできません．またSHELLコマンドの実行後，画面上に以前より残っているBASICのプログラム行やコマンド行に対してスクリーンエディタによる編集を加えても，正しく修正できないことがあります．このようなときには，いちど画面をクリアすればスクリーンエディタは正常状態に戻ります．

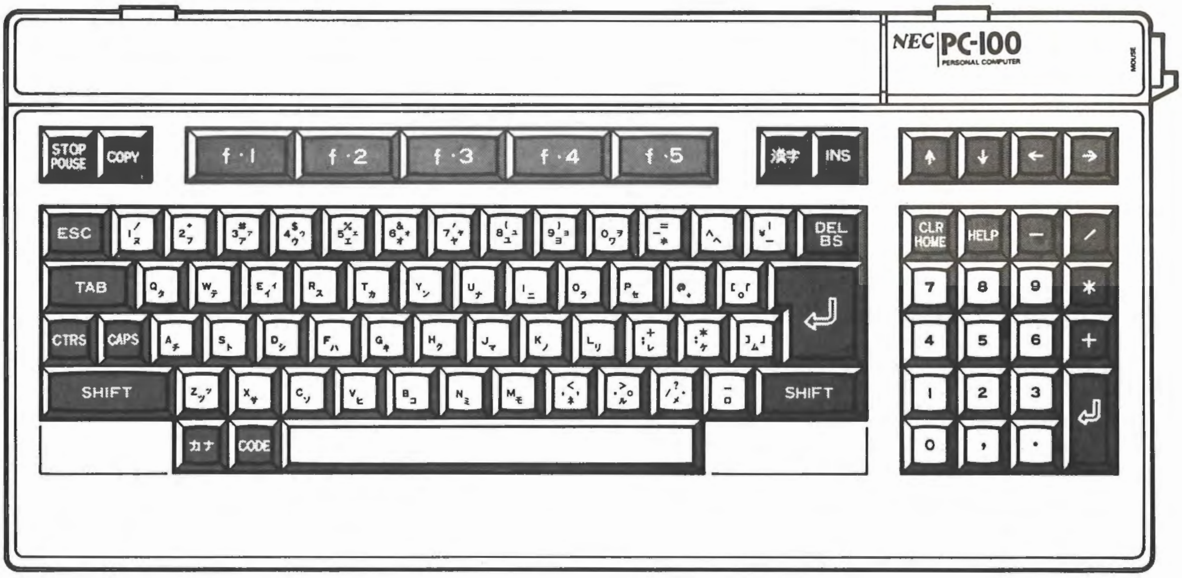
第1章
スクリーンエディタ

N$_{100}$-BASIC では，プログラムを作ったり，コマンドやステートメントをダイレクトに実行させたりするときにスクリーンエディタという画面編集機能を使います．

スクリーンエディタは，ディスプレイ画面上のどの位置にでもカーソルを移動して，入力，修正をすることができます．これによって修正する行を改めて最初から入力する必要がなくなり，プログラムの入力や修正も短時間で行うことができます．

1.1　編集に使うキーの操作

各キーの説明は PC-100ユーザーズガイドに詳しく述べられています．ここでは簡単に表にしてみました．



編集に使うキー

編集キー	機　　能
SHIFT	アルファベットの大文字、キートップの上段に書いてある特殊文字の入力に使用します.
CAPS	アルファベットを大文字で入力するときに使用します.
カ ナ	カタカナの入力に使用します.
CODE	グラフィックキャラクタの入力に使用します.
漢 字	カナ漢字変換モードに入るときに使用します.
STOP PAUSE	SHIFT キーを押しながら押すと、STOP キーになり、プログラムの中断や、行のキャンセルに使います. 単独で押したときは PAUSE キーになり、表示や実行の一時停止に使います.
TAB	カーソルを水平方向に 8 桁ずつ移動します.
CTRL	後述のコントロールコードの入力に使用します.
f･1 ～ f･5	キーの内容をユーザーが定義して使用します.
INS	インサートモードに入るとき、抜けるときに使用します.
DEL BS	SHIFT キーを押しながら押すと、DEL キーになり、カーソルのある部分が 1 文字削除されます. 単独で押すとカーソルの前にある文字が 1 文字削除されます.
⏎	1 行の入力に使用します.
↑ ↓ ← →	カーソルを上下左右に動かします.
CLR HOME	SHIFT キーを押しながら押すと、CLR キーになり、画面のクリアをします. 単独で押すと HOME キーになり、カーソルを画面の左上に移動します.

1.2　ダイレクトモードによる実行

コマンドやステートメントをプログラムにしないで，単独で実行させるモードを**ダイレクトモード**といいます．

ダイレクトモードでの実行は，次の例のように行います．

画面にはコマンド入力待ちのプロンプトが表示されています．

```
Ok
█
```

ここでは例として，ディスプレイ画面上に図形を描くDRAW文を実行させてみましょう．次のようにキー入力してください．[CAPS LOCK]キーが，ロック（押された状態になっていること）されていれば大文字，そうでなければ小文字が入力されますが，どちらでもかまいません．

```
Ok
DRAW "U10"█
```

ここまで入力しただけでは，N_{100}-BASICは何も実行しません．コマンドを実行させるためには，[↵]キーを入力しなければなりません．[↵]キーを押すとどうなるでしょう．

```
Ok
DRAW "U10"
Ok
█
```

DRAW文が実行されて画面に直線が描かれました．

次にスクリーンエディタを使って，DRAW "U10" を DRAW "R10H20" に直して実行してみます．
[↑]キーと[→]キーを使って，カーソルをUの位置まで移動させます．

```
Ok
DRAW"U10"
Ok
```

UをRに変更するには，ここでRを入力するだけでいいのです．

```
Ok
DRAW "R10"
Ok
```

次に H20 を挿入するためにカーソルを文末の" の位置まで移動します．

```
Ok
DRAW "R10"
Ok
```

ここで[INS]キーを押すことによりカーソルの形が変わって，インサートモードになります．

```
Ok
DRAW "R10"_
Ok
```

H20 と入力すると" は順に右にずれます.

```
Ok
DRAW "R10H20"
Ok
```

↵ キーを入力すると，DRAW 文が実行され，カーソルはもとの形に戻ります.

```
Ok
DRAW "R10H20"
Ok
█
```

このように，画面に表示されている文字は修正して何度でも実行させることができます．修正しなくても，文の表示されている行にカーソルを移動して ↵ キーを入力すれば，何回でも実行させることができます.

1.3　プログラムモードによる実行

BASIC のコマンドやステートメントの集まりを続けて実行させるときは，プログラムを作り，RUN コマンドによって実行します．プログラムを入力するモードを**プログラムモード**といい，前述のダイレクトモードとは次の点が異なります.

1. 文の先頭に，0～65529までの数字をつけて入力します.
2. ↵ キーで1行(255文字以内)を入力しますが，実行はされません．実行するには RUN コマンドまたは GOTO 文を使います.
3. ↵ キーによって1行を入力しても "Ok" プロンプトは表示されません.

ここでは，プログラムモードの入力の例として次のような短いプログラムを入力し，変更，追加，削除などの方法を具体的に説明します.

1.3.1 プログラムの新規作成

まず NEW コマンドを実行し，メモリ上にはプログラムが何もない状態にしておいてから次のプログラムを入力してみます．大文字でも小文字でもかまいません．

各行とも⏎キーの入力によってプログラムとしてメモリ上に格納されます（⏎マークは画面上には表示されません）．

```
10 FOR I=1 TO 10 ⏎
20 NEXT I ⏎
30 END ⏎
█
```

プログラムを新しく入力したり，変更したりしたら，必ず LIST コマンドで正しく入力されたかどうかを確かめます．LIST⏎と入力すると次のように表示されます．

```
LIST ⏎
10 FOR I=1 TO 10
20 NEXT I
30 END
Ok
█
```

もしも LIST コマンドによって表示されたプログラムに誤りがあったら，ダイレクトモードの項で述べたようにして訂正します．このときも⏎キーを入力しなければ何も変更したことになりません．プログラムを変更する時には必ず⏎キーを入力しなければなりません．

RUN ⏎ と入力すると，このプログラムが実行されます．

```
RUN ⏎
Ok
█
```

もし，エラーメッセージが表示されたら，LIST コマンドでプログラムを表示し，カーソルを動かしてエラーのあった部分を修正し，⏎ キーを押してプログラムを変更すればいいのです．

1.3.2　行の挿入と追加

BASIC のプログラムは行番号の小さいものから順に実行されます．従って，行番号10と20の間に新しく行を挿入する場合は，11～19の行番号をつけた行を新しく入力すればよいのです．行番号30の後に追加する場合は，31以上の行番号をつけます．

行番号を15，16として次のように入力してみます．

```
15 CIRCLE (300,300),I*10 ⏎
16 BEEP ⏎
```

プログラムの変更を行いましたから，LIST コマンドを実行して確かめてみます．

```
LIST ⏎
10 FOR I=1 TO 10
15 CIRCLE (300,300),I*10
16 BEEP
20 NEXT I
30 END
Ok
█
```

RUN ⏎ と入力して実行させてみましょう．もしエラーが出たらカーソルを誤りのある部分に移動して直し，⏎ キーを押します．

1.3.3　行番号の整理

行をいくつか挿入して行番号が混みあってきたときなどには，RENUM コマンドを使って行番号を整理します．次のように RENUM コマンドを実行し，LIST コマンドでプログラムを表示してみましょう．

```
RENUM 1000 ⏎
Ok
LIST ⏎
1000 FOR I=1 TO 10
1010 CIRCLE (300,300),I*10
1020 BEEP
1030 NEXT I
1040 END
Ok
█
```

RENUM コマンドは，各行番号のあいだの増分や，プログラムのどこから行番号をつけかえるかの指定もできます．詳しくは，N_{100} - BASIC リファレンスマニュアルをご覧ください．

1.3.4　行の複写

ひとつのプログラムの中に，内容が同一の行を何箇所かに作る場合は，行番号の部分だけを書き変えるだけで行のコピーができます．

このプログラムに

```
10 FOR I=1 TO 10
20 NEXT I
```

を追加したい場合は，行番号1000と1030のコピーを作ればよいわけです．カーソルを次図の位置に移動し，BS キーを 2 回押して ⏎ キーを押してください．

```
LIST
1000 FOR I=1 TO 10
1010 CIRCLE (300,300),I*10
1020 BEEP
1030 NEXT I
1040 END
Ok
```

同じように1030を20と書き変えて⏎キーを押します．

```
LIST
10 FOR I=1 TO 10
1010 CIRCLE (300,300),I*10
1020 BEEP
20 NEXT I
1040 END
Ok
```

画面上からは，行番号1000と1030はなくなってしまいましたが，メモリ上には残っています．LIST コマンドを実行してプログラムを表示させてみましょう．

```
LIST ⏎
10 FOR I=1 TO 10
20 NEXT I
1000 FOR I=1 TO 10
1010 CIRCLE (300,300),I*10
1020 BEEP
1030 NEXT I
1040 END
Ok
█
```

1.3.5 行の削除

1行ずつの削除は，行番号だけを入力して ⏎ キーを押します．例えば，

```
1020 ⏎
```

と入力すれば，行番号1020が削除されます．

プログラムのある範囲をまとめて削除する場合は DELETE コマンドが便利です．例えば，次のように行います．

```
DELETE 1000-1010 ⏎
```

プログラム全体を削除するときは NEW コマンドを使用します．

1.4　便利なエディット機能

前の項までで，スクリーンエディタの基本的な使い方について説明しました．ここでは，プログラム作りに慣れたあと，より能率的にスクリーンエディタを使いたいときに便利な機能を紹介します．

1.4.1　コントロールコード

カーソルの移動や文字の削除には，[↑]，[↓]，[←]，[→]，[BS]，[DEL] などのキーを用いてきましたが，他にも次のようなコントロールコードが使用できます．

カーソルの移動

[CTRL]+[B]	カーソルをひとつ前のワードに戻す．
[CTRL]+[F]	カーソルを次のワードに進める．
[CTRL]+[K]	カーソルをホームポジションに移動する．
[CTRL]+[N]	カーソルを行の最後に移す．

文字の削除

[CTRL]+[E]	カーソルのある位置からその行の最後までを削除する．
[CTRL]+[W]	カーソルのあるところから１ワード削除する．
[CTRL]+[Z]	カーソルのある位置以降の画面に表示されている範囲を消去する．

付録のコントコールコード表もあわせてご覧ください．

1.4.2 ロールアップとロールダウン

プログラムが数十行にもわたる場合には，画面上にはその一部しか表示することができませんが，ロールアップ／ロールダウン機能を使うと編集はずっと楽にできます．

この機能は，巻き物を見るように，画面に表示されている行の前後を順に表示させ，修正する部分をさがすために使います．

```
10 ' This is the 1st line.
20 ' This is the 2nd line.
30 ' This is the 3rd line.
40 ' This is the 4th line.
50 ' This is the 5th line.
60 ' This is the 6th line.
70 ' This is the 7th line.
80 ' This is the 8th line.
90 ' This is the 9th line.
100 ' This is the 10th line.
110 ' This is the 11th line.
120 ' This is the 12th line.
130 ' This is the 13th line.
140 ' This is the 14th line.
150 ' This is the 15th line.
160 ' This is the 16th line.
170 ' This is the 17th line.
180 ' This is the 18th line.
190 ' This is the 19th line.
200 ' This is the 20th line.
210 ' This is the 21st line.
220 ' This is the 22nd line.
230 ' This is the 23th line.
240 ' This is the 24th line.
250 ' This is the 25th line.
260 ' This is the 26th line.
270 ' This is the 27th line.
280 ' This is the 28th line.
290 ' This is the 29th line.
300 ' This is the 30th line.
310 ' This is the 31st line.
320 ' This is the 32nd line.
330 ' This is the 33th line.
340 ' This is the 34th line.
350 ' This is the 35th line.
360 ' This is the 36th line.
370 ' This is the 37th line.
380 ' This is the 38th line.
390 ' This is the 39th line.
400 ' This is the 40th line.
```

ロールアップ

ロールダウン

```
100 ' This is the 10th line.
110 ' This is the 11th line.
120 ' This is the 12th line.
130 ' This is the 13th line.
          :
310 ' This is the 31st line.
320 ' This is the 32nd line.
330 ' This is the 33th line.
340 ' This is the 34th line.
```

図の100行の位置にカーソルを移動して SHIFT +↑ キーを押すと100行の前にある行が順に表示されていきます．これがロールアップ機能です．このようにして修正が必要な行をさがしていけばよいわけです．

カーソルが画面の中程にある時はそれより上に表示されている行はそのままで，カーソルの上に，行が順に挿入されていきます．

```
100 ' This is the 10th line.
110 ' This is the 11th line.
120 ' This is the 12th line.
130 ' This is the 13th line.
          :
310 ' This is the 31st line.
320 ' This is the 32nd line.
330 ' This is the 33th line.
340 ' This is the 34th line.
```

図のように画面の最終行にカーソルを移動して SHIFT +↓ キーを押すと，後の行が順に表示されていきます．これがロールダウン機能です．

カーソルが画面の中程にある場合はそれより上に表示されている行はそのままで，カーソルの下に行が挿入されていきます．

1.4.3　EDIT コマンド

プログラムを実行させてエラーが出たとき，EDIT コマンドを使ってエラーの発生した行を表示し，その先頭にカーソルを移動することができます．

例えばプログラムを実行して次のようになったとき，

```
RUN ⏎
Illegal function call in 100
Ok
█
```

EDIT　．⏎ と入力すると，次のように表示されます．

```
EDIT .⏎
100 CIRCLE(X,Y),R
```

この行の前後を表示するときは前述のロールアップ／ロールダウン機能を使います．

1.5　スクリーンエディタ使用上の注意

- プログラムを修正したときは必ず[⏎]キーを押さなければいけません．キャリッジリターンコードは画面には表示されないので注意してください．
- プログラムを変更したあとは，必ずLISTコマンドを実行して確かめる習慣をつけましょう．
- 行は255文字までの長さで作れます．それ以上入力しても無視されてしまいます．
- スクリーンエディタ中の行末に[→]キーや[TAB]キーによる空白があるときに[BS]キーを押すと，空白とその直前の1文字が削除されます．
例えば，

```
10 PRINT[→][→][→][TAB]     █
```

（画面上には行末に空白がある）

の状態で[BS]キーを押すと，

```
10 PRIN█
```

となります．

行末にスペースコードがあるとき，例えば，

```
10 PRINT␣␣␣█
```

（␣はスペースコードを表す）

のとき BS キーを押すと，

```
10 PRINT␣␣█
```

のようにスペースコードがひとつ削除されます．

- N_{100}-BASIC では，ダイレクトモード，プログラムモードで画面にエスケープシーケンスを送ると，スクリーンエディタが正常な動作をしなくなることがありますので注意してください．
  これが原因でスクリーンエディタがおかしな動作をするときは，WIDTH 文，WIDTH USING 文，または SCREEN 文で現在設定されているモードと異なるモードに再設定することによって直すことができます．
  この方法でも直らない場合は，SYSTEM コマンドで MS-DOS に戻らなければなりません．
- SHELL 文によって，N_{100}-BASIC 中から MS - DOS のコマンドを実行することができます．しかし，SHELL 文によって画面上に表示された文字はスクリーンエディタに用いることはできません．また，SHELL 文によって画面がスクロールした場合は画面全体がスクリーンエディットできなくなります．この場合は，CLS 文などで画面をクリアした後，LIST コマンドで再度プログラムを表示してからスクリーンエディットしてください．
- 漢字は 2 バイトで表され，画面上にも英数字 2 文字分のスペースをとって表示されます．

第2章
文字フォントの切換え

PC-100 の MS-DOS では，アプリケーションプログラムで 3 種類の文字フォントを選択できます．N_{100}-BASIC でも，これらの文字フォントを簡単な方法で使用することができます．

文字フォントは次の 3 種類から選択して使用します．

標準体
ボールド体
イタリック体

このうち標準体のフォントセットは ROM に用意されており，何の手続きもなしに使用できます．

ボールド体とイタリック体は，MS-DOS システムディスクに用意されている BOLD. SYS, ITALIC. SYS というフォントセットファイルを，MS-DOS の起動時にメモリ上に読み込んでおかなくてはなりません．この方法については MS-DOS ユーザーズガイドの付録をご覧ください．

上記の手続きを行い，MS-DOS の起動時にフォントセットファイルを読み込んであるとき，N_{100}-BASIC では次のようにキー入力することによって文字フォントの切換えを行うことができます．

ボールド体に切り換えるとき

```
PRINT CHR$(27)+ "[1m"
```

イタリック体に切り換えるとき

```
PRINT CHR$(27)+ "[3m"
```

標準体に切り換えるとき

```
PRINT CHR$(27)+ "[0m"
```

注意：上記の文字フォントの切換えを行う場合は、8 ×16ピクセルの文字フォントを選択しなくてはなりません（N_{100}-BASIC リファレンスマニュアル の WIDTH 文，WIDTH USING 文参照）．

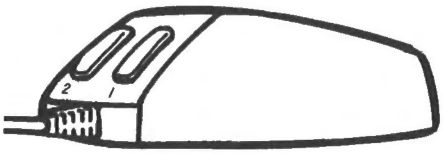
第3章
マウス

マウスはとても強力な働きをしてくれます．めんどうなキーボード操作からユーザーを解放してくれますし，独特の関数により画面処理がとても楽にできます．

あなたも，マウスを自分の手足として使いこなしてください．

3.1　マウスとは

マウスは下にボール，正面に2つのボタンがつき，ちょうどねずみのような形をしています．



側　面

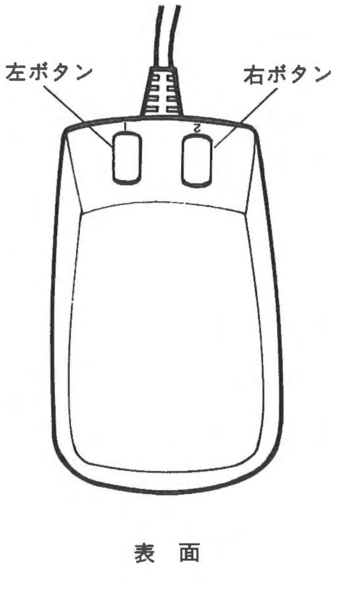
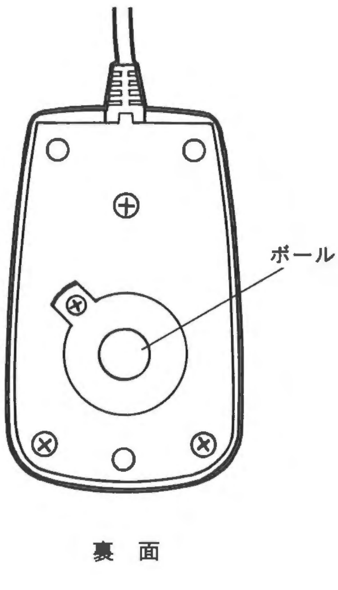
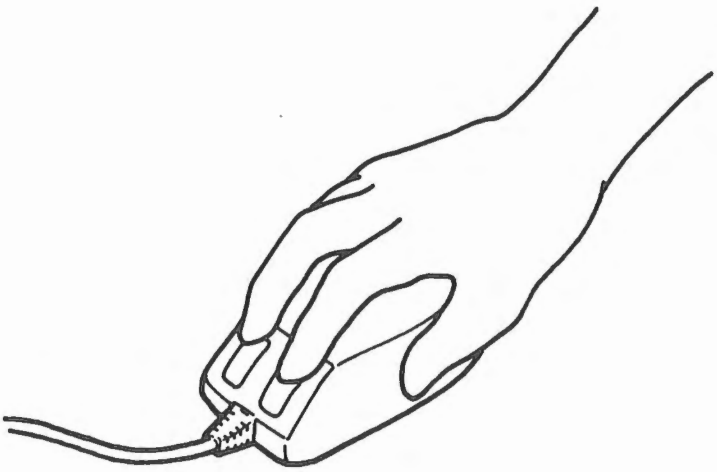


表　面

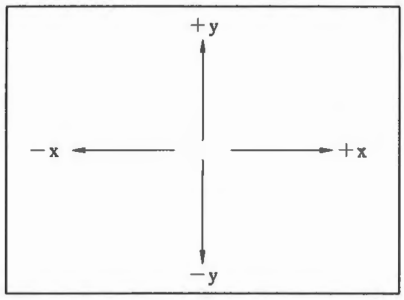


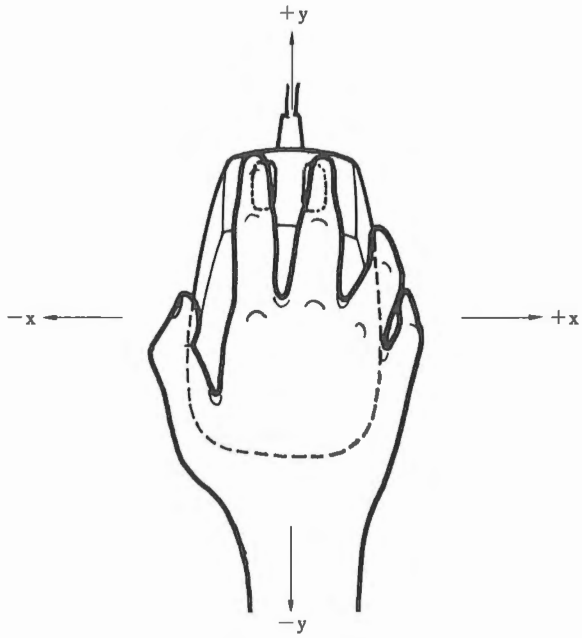
裏　面

マウスは，正面を向こう側に向けてにぎり，二本の指で，２つのボタンに触れるようにして持ってください。

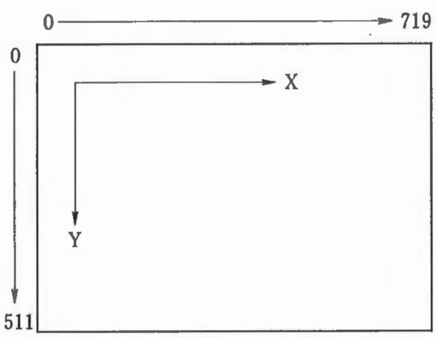


マウスを平らな盤上で動かすと，ボールが回転して画面上でマウスカーソルが移動します．このときマウスカーソルはマウスの移動方向と対応した方向に動きます．





マウスカーソルの移動範囲は画面が横表示の場合，横720ピクセル，縦512ピクセルです．



また，マウスには1（左側）と2（右側）の2つのボタンがついています．これらを操作することにより，さまざまなマウス関数の値（例えば，カーソルの座標など）をとり出したり，マウス割込みで割込み処理を行ったりできます．

3.2　マウスのステートメント

3.2.1　初期設定

マウスの使用前，使用後には必ず，

MOUSE 0

を実行してください．これにより，マウスステートメントで設定された諸機能（カーソルの形状・移動比率･移動範囲など)を初期状態に戻します．例えば，カーソルの形状は矢印に戻され，移動範囲は(0，0)—(719,511)に戻されます．そして，カーソルの表示をとりやめ，マウスの機能を完全にOFF(停止)状態にします．

3.2.2　カーソルの位置設定

マウスカーソルの表示／非表示と，それらの位置を指定する場合

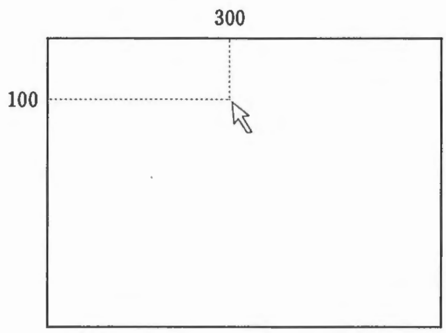
MOUSE　1, x, y,<カーソルスイッチ>

を使ってください．指定した座標(x, y)でマウスを表示／非表示できます．<カーソルスイッチ>は0で非表示，1で表示となります．

たとえば，

MOUSE 1, 300, 100, 1 ⏎

とすると，マウスカーソルが（300,100）の位置に表示されます．



マウスを動かすと，カーソルの移動を見ることができます．

ところで，パラメータ x と y は省略することもできます．以下をダイレクトモードで実行してみてください．

MOUSE 0 ⏎
MOUSE 1, 300, 100, 1 ⏎
↓
カーソルを適当に移動させる．
↓
MOUSE 1, , , 0 ⏎……カーソルはその場所で非表示となる．
MOUSE 1, , , 1 ⏎……非表示した場所に表示される．
MOUSE 1, , , 0 ⏎
↓
カーソルを適当に移動させる．
↓
MOUSE 1, , , 1 ⏎
MOUSE 1, 300, 100, 0 ⏎……(300,100) の位置にカーソルは移動するが非表示となっている．
MOUSE 1, , , 1 ⏎……(300,100) に表示される．

3.2.3　移動範囲の設定

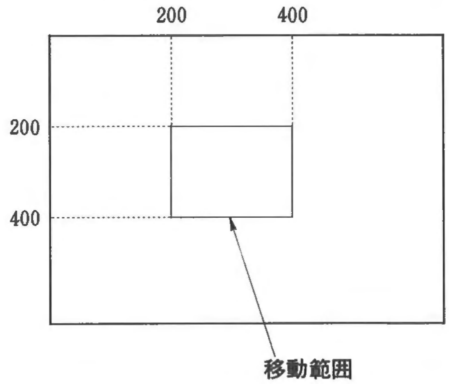
初期状態におけるマウスカーソルの移動範囲は，画面が横表示の場合，X方向が0～719，Y方向が0～511です．画面が縦表示の場合はX方向が0～511，Y方向が0～719となります．この移動範囲は，下記のステートメントで，自由に変更できます．

MOUSE 4, x_1, y_1, x_2, y_2,

これを実行することにより，2点 (x_1，y_1) と (x_2，y_2) を対角点とする四角形の内部が，マウスカーソルの移動範囲として設定されます．

ダイレクトモードで以下を実行してみてください．

MOUSE 0 ⏎
MOUSE 1, 0, 0, 1 ⏎
MOUSE 4, 200, 200, 400, 400 ⏎
↓
マウスを移動させてみる．



ここで LINE(200, 200)－(400, 400), 15, B ⏎ とすると，マウスの移動範囲が設定されているのがよくわかります．

この状態でつぎの文を実行してみます．

MOUSE 1, 700, 500, 1 ⏎

おかしいと思いませんか？　最後で，カーソルを (700,500) に表示したはずなのに，全く違った場所に表示されています．これは，指定された座標が移動範囲外の場合，カーソルは設定された範囲内の最も近い点に表示されるようになっているからです．また，

MOUSE 4,100,100,100,100

とした場合，カーソルは(100,100) に固定されることになります．
なお，このステートメントのパラメータを省略することはできません．

3.2.4　移動速度の設定

マウスカーソルの移動速度を変更したい場合は，次のステートメントで設定してください．

MOUSE 3, 〈方向〉, 〈比率〉

〈方向〉はX方向には0，Y方向には1を指定します．〈比率〉とは，マウスの移動距離に対するカーソルの移動距離の割合いを意味しています．この値は0～32767の範囲で，大きい程移動速度が遅くなります．初期設定において，この値はX方向，Y方向とも0となっています．

ダイレクトモードで実行してみましょう．

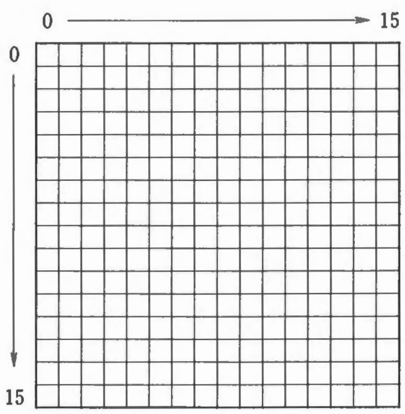
MOUSE 0: MOUSE 1, 0, 0, 1
MOUSE 3, 0, 200
↓
カーソルを動かしてみる．
X方向の移動速度が遅くなっているのがわかる．
↓
MOUSE 3, 1, 400
↓
カーソルを動かすと，Y方向の移動速度も遅くなっている．

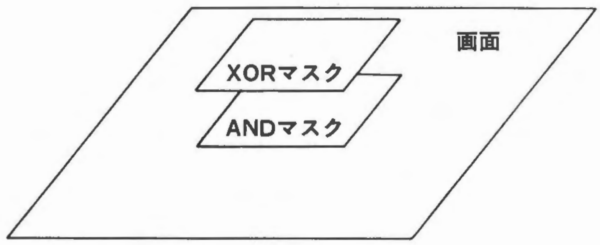
3.2.5　カーソルの形状

マウスカーソルの形状は，次のステートメントを使用することにより，任意の形に変えることができます．

MOUSE 2, 〈AND マスク〉, 〈XOR マスク〉, X_c , Y_c

カーソルパターンは，図のように16×16ピクセルで示されます．そして，カーソルの形状はANDマスク・XORマスクの2つのマスクで表わされます．





ANDマスクとXORマスクの関係

AND	XOR	ドットパターン	画面上
1	0	地のまま	run
1	1	地の反転	run
0	0	黒	n
0	1	白	n

ステートメント中のパラメータ，X_cとY_cは，〈X指示点〉と〈Y指示点〉で，それぞれの0〜15の範囲で指定します．マウス関数などで用いられる座標は，この指示点を参照するのです．

カーソルの初期状態を図示しておきますので，よくわからない人は，図をじっくりと見てください．データは図のように2進表現となっています．これを16進に変換して使用します．

例） 計算のしかた

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	0	1	0	0	1	1	1	0	0	1	1	0

⇓

01110100　　　　11100110　　　　← 2進表記

⇓

&H74　　　　&HE6　　　　←16進表記

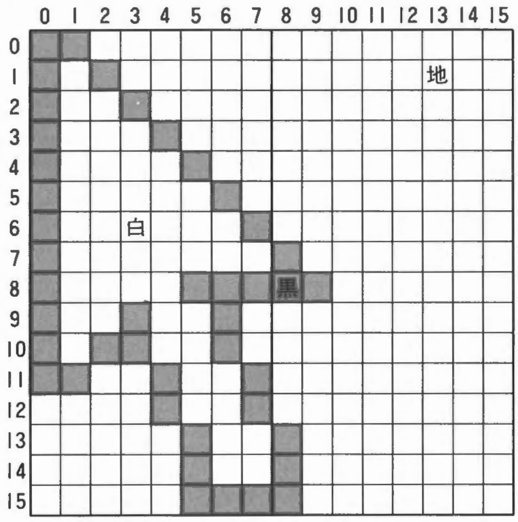
ANDマスク																データ	
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3F	FF
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1F	FF
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0F	FF
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	07	FF
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	03	FF
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	01	FF
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	FF
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	7F
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	3F
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	01	FF
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	01	FF
0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	30	FF
1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	F0	FF
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	F8	7F
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	F8	7F
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	F8	7F

XORマスク

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

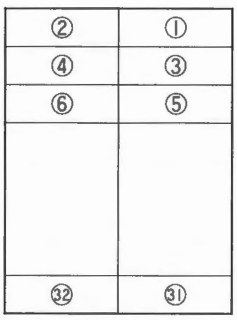
データ

0	0
40	0
60	0
70	0
78	0
7C	0
7E	0
7F	0
78	0
6C	0
4C	0
06	0
06	0
03	0
03	0
0	0



ANDマスクとXORマスクにより，左図のように見えるのです．
初期の指示点は（0，0）となっています．

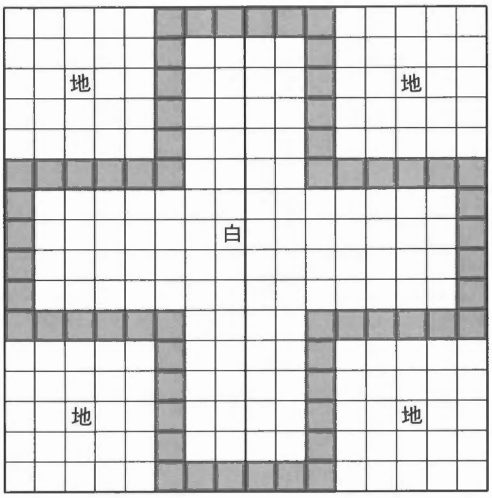
次に，マスクの設定方法を説明します．マスクは，32文字（32×8ビット）を図のような順序で継げることにより，設定できます．初期設定のデータを例にとって説明しておきます．



①+②+③+④+……+㉛+㉜

```
A$=CHR$(&HFF, &H3F, &HFF, &H1F, ……, &HF8, &H7F, &HF8)  ……ANDマスク
X$=CHR$ (&H0, &H0, &H0, &H40, ……, &H0, &H03, &H0, &H0)……XORマスク
MOUSE 2, A$, X$, 0, 0
```

次のプログラムを実行すると，十字のカーソルが表示されます．



ANDマスク		XORマスク	
F8	1F	0	0
F8	1F	3	C0
F8	1F	3	C0
F8	1F	3	C0
F8	1F	3	C0
0	0	3	C0
0	0	7F	FE
0	0	7F	FE
0	0	7F	FE
0	0	7F	FE
0	0	3	C0
F8	1F	3	C0
F8	1F	3	C0
F8	1F	3	C0
F8	1F	3	C0
F8	1F	0	0

```
10 MOUSE 0:SCREEN 1,0,0,0:CLS
20 A$=CHR$(&H1F,&HF8,&H1F,&HF8,&H1F,&HF8,&H1F,&HF8,&H1F,&HF8
):B$=CHR$(0,0,0,0,0,0,0,0,0,0,0,0):C$=CHR$(&HC0,3,&HC0,3,&HC
0,3,&HC0,3,&HC0,3):D$=CHR$(&HFE,&H7F,&HFE,&H7F,&HFE,&H7F,&HF
E,&H7F)
40 MOUSE 2,A$+B$+A$,CHR$(0,0)+C$+D$+C$+CHR$(0,0)
50 LINE(0,0)-(719,511),12,BF
60 MOUSE 1,360,255,1
```

3.3 マウス関数

マウスステートメントだけでは，マウスを使いこなすことはできません．マウス関数を使うことにより，プログラム中ですばらしい力を発揮してくれます．

3.3.1 現在のカーソル座標

現在のカーソル座標を示すのは，次の関数です．

MOUSE (0)……現在のX座標の値を返す．

MOUSE (1)……現在のY座標の値を返す．

次のプログラムを実行すると，カーソルの動きにそって線が描かれます．また，左上に現在の座標を表示します．

```
10 SCREEN 1,0,0,0:MOUSE 0:CLS
20 MOUSE 1,0,0,1:POINT(0,0)
30 X=MOUSE(0):Y=MOUSE(1)
40 LOCATE 1,1,0:PRINT"X=";X;"Y=";Y
50 MOUSE 1,,,0:LINE-(X,Y):MOUSE 1,,,1
60 GOTO 30
```

3.3.2　ボタンの状態

ボタンが押されているか，押されていないかという状態を調べるには，次の関数を使います．なお，ボタン番号は1が左のボタンで2が右のボタンです．

MOUSE (2, 〈ボタン番号〉)

指定されたボタンが押されている場合は1，押されてない場合は0を返します．

次のプログラムを実行してみましょう．画面にカーソルは表示されませんが，ボタンを押せば音が出ます．画面左上には，それぞれのボタンの状態が表示されます．

```
10 MOUSE 0:CLS 3
20 B1=MOUSE(2,1):B2=MOUSE(2,2)
30 LOCATE 1,1,0:PRINT "LEFT=";B1,"RIGHT=";B2
40 IF B1=1 THEN PLAY"T255C20"
50 IF B2=1 THEN PLAY"T255A20"
60 GOTO 20
```

3.3.3　ボタンとカーソルの座標

ボタンを押したときや，離したときの座標を調べるには，次の関数を使います．

MOUSE (4, 〈ボタン番号〉)…最後にボタンが押されたときのX座標の値を返す．

MOUSE (5, 〈ボタン番号〉)…最後にボタンが押されたときのY座標の値を返す．

MOUSE (7, 〈ボタン番号〉)…最後にボタンが離されたときのX座標の値を返す．

MOUSE (8, 〈ボタン番号〉)…最後にボタンが離されたときのY座標の値を返す．

次のプログラムでは，MOUSE(4,　)とMOUSE(5,　)を使っています．左ボタンを押すと，直前に押された座標から現在の位置まで線を引きます．右ボタンを押すと終了します．

```
10 SCREEN 1,0,0,0:MOUSE 0
20 CLS 3:MOUSE 1,0,0,1:POINT(0,0)
30 B1=MOUSE(2,1):B2=MOUSE(2,2)
40 IF B2=1 THEN MOUSE 0:END
50 IF B1=0 THEN 30
60 X1=MOUSE(4,1):Y1=MOUSE(5,1)
70 MOUSE 1,,,0:LINE-(X1,Y1):MOUSE 1,,,1
80 GOTO 30
```

次のプログラムではMOUSE(7,　)とMOUSE(8,　)を使っています．左ボタンを押しながらマウスを移動させ，ボタンを離すと，押した点から離した点まで線を引きます．右ボタンを押すと終了します．

```
10 SCREEN 1,0,0,0:MOUSE 0
20 CLS 3:MOUSE 1,0,0,1:POINT(0,0)
30 B1=MOUSE(2,1):B2=MOUSE(2,2)
40 IF B2=1 THEN MOUSE 0:END
50 IF B1=0 THEN 30
60 IF MOUSE(2,1)=1 THEN 60
70 X1=MOUSE(7,1):Y1=MOUSE(8,1)
80 MOUSE 1,,,0:LINE-(X1,Y1):MOUSE 1,,,1
90 GOTO 30
```

3.3.4 ボタン操作の回数

ボタンを押した回数や離した回数を調べるには，次の関数を使用します．

MOUSE (3, 〈ボタン番号〉)…１度この関数を呼び出してから次に呼び出すまでに，ボタンの押された回数を返す．

MOUSE (6, 〈ボタン番号〉)…上と同様で，ボタンが離された回数を返す．

ダイレクトモードで実行してみましょう．

MOUSE 0 ⏎

PRINT MOUSE(3, 1) ⏎

↓

0が表示される．

左ボタンを５回押してみる．

↓

PRINT MOUSE(3, 1) ⏎

↓

5が表示される．

MOUSE(6,)に関しても同じです．

3.3.5 移動距離

カーソルが動いたピクセル数を調べるには，次の関数を使います．

MOUSE (9)…１度この関数を呼び出してから，次に呼び出すまで，カーソルが動いたＸ方向の移動距離を返す．

MOUSE (10)…上と同じように，Ｙ方向の移動距離を返す．

ダイレクトモードで実行してみましょう．

MOUSE 0

PRINT MOUSE(9); MOUSE(10)

↓

カーソルを移動させる．

↓

PRINT MOUSE(9) MOUSE(10)

↓

Ｘ方向，Ｙ方向のそれぞれの移動距離が表示される．

3.4　マウス割込み

マウスの操作に対応して割込みを起こさせ，特定の処理を行わせることができます．

番号　N	対応する操作
1	カーソル移動
2	左ボタンを押した
3	右ボタンを押した
4	左ボタンを離した
5	右ボタンを離した

MOUSE（N）ON／OFF／STOP

上の命令は，番号Nに対応する操作での，割込みを指定します．

ON………割込みを許可
OFF……割込みを禁止
STOP……割込みを保留

割込みが起きた場合，次のステートメントで設定された処理ルーチンの開始行へ分岐します．

ON MOUSE(N) GOSUB<行番号>

このステートメントのNは，前の表を参照してください．
次のプログラムは，すべてのマウス操作に対応して，それぞれ違う音が出ます．

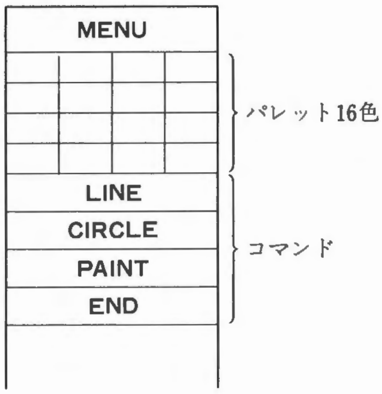
```
100 MOUSE 0:CLS:MOUSE 1,0,0,1
110 FOR I=1 TO 5:MOUSE (I) ON:NEXT
120 ON MOUSE (1) GOSUB 180
130 ON MOUSE (2) GOSUB 190
140 ON MOUSE (3) GOSUB 200
150 ON MOUSE (4) GOSUB 210
160 ON MOUSE (5) GOSUB 220
170 GOTO 170
180 PLAY"T100c10":RETURN
190 PLAY"T255C10":RETURN
200 PLAY"T255D10":RETURN
210 PLAY"T255E10":RETURN
220 PLAY"T255F10":RETURN
```

3.5 応用プログラム

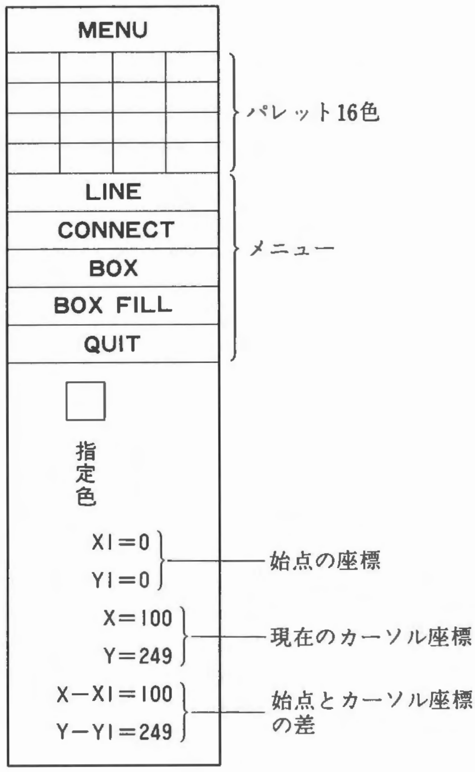
マウスをクレヨンに，画面を画用紙にして絵を描くプログラムです．画家になったつもりで絵を描いてください．

それでは，使用方法を説明しましょう．

1. プログラムを実行すると，図のようなメニューが表示されます．実行したいコマンド（線・円・色塗り）の上にカーソルを移動して，左ボタンを押してください．
   END を選ぶと，プログラムは終了します．



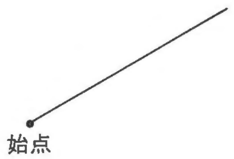
2. LINE コマンド
   LINE を選ぶと，図のようにメニューが変わりますので，左ボタンで線の色と線の種類を選んでください．このとき，カーソルの形が，えんぴつに変化します．現在選んでいる色は，メニューの下部の〝指定色〞に表示されます．またその下の，X1 と Y1 は始点（描き始め）の座標で，X と Y はカーソルの現在の座標を示しています．

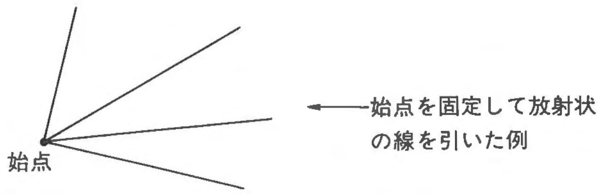


次にカーソルを動かして左のボタンを押すと，そこが始点となります．右のボタンでもう 1 点を指定すると，線を描きます．

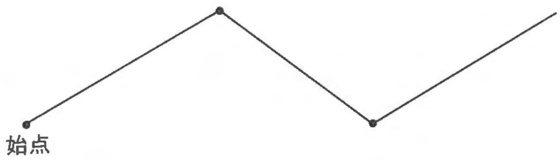
線の種類

LINE ……………BASICのLINE文と同じで線を引きます．始点は変更しない限り，同じ点を使います．また，メニュー下のX－X1の値を0にすれば垂直線が，Y－Y1の値を0にすれば，水平線が引けます．

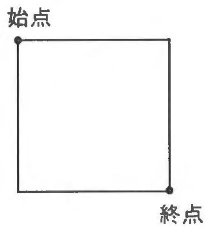




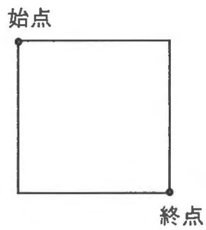
CONNECT ……一筆書きのように，つながった線を引きます．左のボタンで始点を指定してから，右のボタンを押すだけでどんどん線を描いてくれます．



BOX ……………長方形を描く．



BOX FILL ……塗りつぶした長方形を描く．

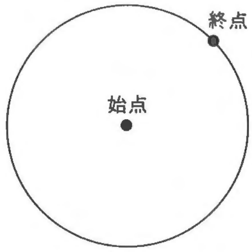


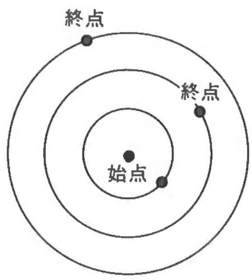
QUIT を選ぶと，初期のメニューに戻ります．

3.　CIRCLE コマンド

このコマンドを選ぶと，カーソルがコンパスに変化します．まず色を選んでください．それから，左のボタンで円の中心を，右のボタンで円周上の1点を指定すると，円を描きます．始点を固定した場合，同心円が描けます．

QUIT で初期のメニューに戻ります．

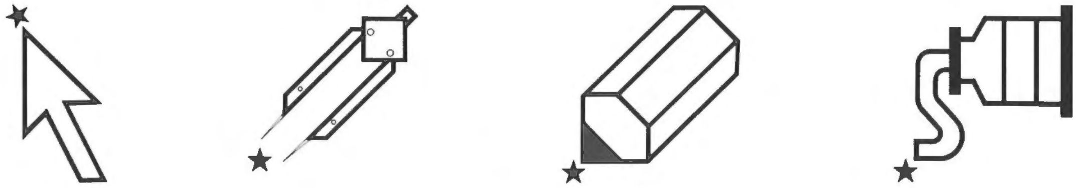




4. PAINT コマンド

このコマンドを選ぶと，カーソルが絵の具のチューブに変化します．左のボタンで塗りたい色を，右のボタンで境界色を選んでください．それぞれの色は，メニューの下に表示されます．カーソルを塗りたい場所に移動させ，左ボタンを押すとペイントしてくれます．

注）それぞれのカーソルの示す座標は次の図の星印の点です．



プログラムリスト

```
100 '*********************************
110 '***                          ***
120 '***    OEKAKI プログラム      ***
130 '***                          ***
140 '*********************************
150 '***** INITIALIZE *****
160 WIDTH 90,25:SCREEN 1,0,0,0:MOUSE 0:CLS:DEFINT X,Y
170 LA$=CHR$(&H7,&HFE,3,&HFC,1,&HF8,0,&HF0,0,&HE0,0,&HC0,0,&
H80,0,0,0,0,1,0,3,0,7,0,&HF,0,&H1F,0,&H3F,0,&H7F,0)
180 LX$=CHR$(0,0,&HF0,0,&H28,1,&H44,2,&H82,4,6,9,&HA,&H12,&H
12,&H24,&H24,&H58,&H48,&H44,&H90,&H46,&H20,&H61,&H40,&H71,&H
80,&H7C,0,&H7F,0,0)
190 CA$=CHR$(&HF8,&HFF,&H80,&HFF,0,&HFE,1,&HFC,1,&HF8,1,&HF0
,1,&HE0,3,&HC0,3,&HC0,7,&HC0,&HF,&H80,&H1F,0,&H3F,0,&H7F,0,&
HFF,&HC3,&HFF,&HC7)
200 CX$=CHR$(0,0,2,0,&H3C,0,&HDC,0,&HBC,1,&HB4,3,8,7,&H38,&H
E,&H70,&H14,&HE0,&H18,&HC0,&H11,&H80,&H22,0,&H47,0,8,0,&H10,
0,0)
210 PA$=CHR$(&HF8,&HFF,0,&HFC,0,&HE0,0,&HE0,0,&H80,0,0,0,0,0
,0,0,0,0,0,0,0,&HF8,&H81,&HFF,1,&HFF,1,&HFF,1,&HFF,3 )
220 PX$=CHR$(0,0,0,2,&HFE,1,&H4A,&HA,&H4A,&HE,&H4A,&H28,&H4A
,&H68,&H4A,&H6E,&H4A,&H6A,&HFE,&H61,2,&H30,0,&H18,0,&HC,0,&H
6C,0,&H78,0,0)
230 LINE(640,0)-(640,712)
240 LOCATE 1,82:PRINT"MENU"
250 FOR I=17 TO 97 STEP 20
260 LINE(640,I)-(720,I)
270 NEXT I
280 FOR I=660 TO 720 STEP 20
290 LINE(I,17)-(I,97)
300 NEXT I
310 FOR I=645 TO 715 STEP 20
320 FOR J=27 TO 87 STEP 20
330 PAINT(I,J),P,15:P=P+1
340 NEXT J:NEXT I
```

```
350 '***** COMMAND *****
360 MOUSE 0:LINE(641,98)-(719,511),0,BF
370 LOCATE 6,82:PRINT"LINE"
380 LOCATE 7,82:PRINT"CIRCLE"
390 LOCATE 8,82:PRINT"PAINT"
400 LOCATE 9,82,0:PRINT"END"
410 FOR I=117 TO 177 STEP 20
420 LINE(640,I)-(720,I)
430 NEXT I
440 MOUSE 1,X,Y,1
450 X=MOUSE(0):Y=MOUSE(1)
460 IF MOUSE(2,1)=0 THEN 450
470 IF X<640 OR Y<97 OR Y>177 THEN BEEP:GOTO 450
480 IF Y<117 THEN GOSUB 530:GOTO 360
490 IF Y<137 THEN GOSUB 880:GOTO 360
500 IF Y<157 THEN GOSUB 1100:GOTO 360
510 FOR I=0 TO 2000:PALETTE I MOD 16,RND*512:FOR J=0 TO 100:
NEXT J:NEXT I:PALETTE:END
520 '***** LINE *****
530 MOUSE 0:LINE(641,98)-(719,511),0,BF
540 LOCATE 6,82:PRINT"LINE"
550 LOCATE 7,82:PRINT"CONNECT"
560 LOCATE 8,82:PRINT"BOX"
570 LOCATE 9,82:PRINT"BOX FILL"
580 LOCATE 10,82,0:PRINT"QUIT"
590 FOR I=117 TO 197 STEP 20
600 LINE(640,I)-(720,I)
610 NEXT I
620 GOSUB 1300:GOSUB 770
630 MOUSE 2,LA$,LX$,1,16
640 MOUSE 1,X,Y,1
650 X=MOUSE(0):Y=MOUSE(1):LOCATE 19,84:PRINT "X=";X:LOCATE 2
0,84:PRINT "Y=";Y:LOCATE 22,82:PRINT USING "X-X1= ###";ABS(X
-X1):LOCATE 23,82:PRINT USING "Y-Y1= ###";ABS(Y-Y1);
660 IF MOUSE(2,2)=1 THEN 780
670 IF MOUSE(2,1)=0 THEN 650
680 IF X>640 THEN 700
690 X1=X:Y1=Y:LOCATE 16,85:PRINT USING "= ###";X1:LOCATE 17,
85:PRINT USING "= ###";Y1:GOTO 650
700 IF Y>197 OR Y<17 THEN BEEP:GOTO 650
710 IF Y>177 THEN RETURN
720 IF Y>157 THEN BL=NOT BL:BX=0:CO=0:LI=0:GOSUB 770:GOTO 65
0
730 IF Y>137 THEN BX=NOT BX:BL=0:CO=0:LI=0:GOSUB 770:GOTO 65
0
740 IF Y>117 THEN CO=NOT CO:BL=0:BX=0:LI=0:GOSUB 770:GOTO 65
0
750 IF Y>97  THEN LI=NOT LI:BL=0:BX=0:CO=0:GOSUB 770:GOTO 65
0
760 C=POINT(X-2,Y-2):GOSUB 1320:GOTO 650
770 MOUSE 1,X,Y,0:PAINT(641,138),BX*-1,15:PAINT(641,118),CO*
-1,15:PAINT(641,98),LI*-1,15:PAINT(641,158),BL*-1,15:MOUSE 1
,X,Y,1:RETURN
780 IF X>640 THEN BEEP:GOTO 650
790 X2=X:Y2=Y
```

```
800 MOUSE 1,X,Y,0
810 VIEW(0,0)-(639,511)
820 IF BX THEN LINE(X1,Y1)-(X2,Y2),C,B:GOTO 860
830 IF BL THEN LINE(X1,Y1)-(X2,Y2),C,BF:GOTO 860
840 LINE(X1,Y1)-(X2,Y2),C
850 IF CO THEN X1=X2:Y1=Y2:LOCATE 16,85:PRINT USING "= ###";
X1:LOCATE 17,85:PRINT USING "= ###";Y1
860 VIEW:MOUSE 1,X2,Y2,1:GOTO 640
870 '***** C I R C L E *****
880 MOUSE 0
890 LINE(641,98)-(720,511),0,BF
900 LOCATE 6,82,0:PRINT "QUIT"
910 LINE(640,117)-(720,117)
920 GOSUB 1300
930 MOUSE 2,CA$,CX$,1,16
940 MOUSE 1,X,Y,1
950 X=MOUSE(0):Y=MOUSE(1):LOCATE 19,84:PRINT "X=";X:LOCATE 2
0,84:PRINT "Y=";Y:LOCATE 22,82:PRINT USING "X-X1= ###";ABS(X
-X1):LOCATE 23,82:PRINT USING "Y-Y1= ###";ABS(Y-Y1);
960 IF MOUSE(2,2)=1 THEN 1030
970 IF MOUSE(2,1)=0 THEN 950
980 IF X>640 THEN 1000
990 X1=X:Y1=Y:LOCATE 16,85:PRINT USING "= ###";X1:LOCATE 17,
85:PRINT USING "= ###";Y1:GOTO 950
1000 IF Y<17 OR Y>117 THEN BEEP:GOTO 950
1010 IF Y>97 THEN RETURN
1020 C=POINT(X-2,Y-2):GOSUB 1320:GOTO 950
1030 IF X>640 THEN BEEP:GOTO 980
1040 X2=X:Y2=Y
1050 MOUSE 1,X,Y,0
1060 VIEW(0,0)-(639,511)
1070 CIRCLE(X1,Y1),SQR((X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)),C,,,
.99
1080 VIEW:MOUSE 1,X2,Y2,1:GOTO 940
1090 '***** P A I N T *****
1100 MOUSE 0:LINE(641,98)-(720,511),0,BF
1110 LOCATE 6,82,0:PRINT"QUIT"
1120 LINE(640,117)-(720,117)
1130 GOSUB 1310:GOSUB 1330
1140 MOUSE 2,PA$,PX$,1,16
1150 MOUSE 1,X,Y,1
1160 X=MOUSE(0):Y=MOUSE(1)
1170 CC=POINT(X-2,Y-2)
1180 IF X<640 THEN 1240
1190 IF Y<17 OR Y>117 THEN 1160
1200 IF MOUSE(2,2)=1 THEN C1=CC:GOSUB 1340:GOTO 1160
1210 IF MOUSE(2,1)=0 THEN 1160
1220 IF Y>97 THEN RETURN
1230 C=CC:GOSUB 1320:GOTO 1160
1240 IF MOUSE(2,1)=0 THEN 1160
1250 MOUSE 1,X,Y,0
1260 VIEW(0,0)-(639,511)
1270 PAINT(X,Y),C,C1
1280 VIEW:GOTO 1150
```

```
1290 '***** COLOR 表示 *****
1300 LOCATE 16,83:PRINT"X1=";X1:LOCATE 17,83:PRINT"Y1=";Y1
1310 LOCATE 13,82:PRINT"指":LOCATE 14,82:PRINT"定":LOCATE 15
,82:PRINT"色"
1320 LINE(650,210)-(670,240),15,B:LINE(651,211)-(669,239),C,
BF:RETURN
1330 LOCATE 13,87:PRINT"境":LOCATE 14,87:PRINT"界":LOCATE 15
,87:PRINT"色"
1340 LINE(690,210)-(710,240),15,B:LINE(691,211)-(709,239),C1
,BF:RETURN
```

第4章
グラフィックス

4.1　画面について

N_{100}-BACIC は，SCREEN 文を使って画面モード，画面ポジションを選択することができます．また，N_{100}-BACIC はリストなどのキャラクタを出力する画面と，点や線などのグラフィックスを出力する画面を別の画面として設定することはできません．

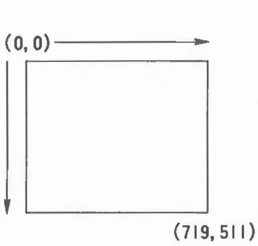
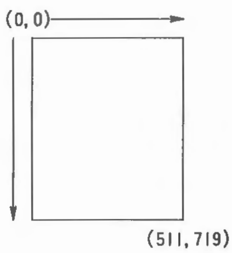
4.1.1　画面モード

画面モードには，白黒モードとカラーモードがあります．この指定は，SCREEN 文の第１パラメータで行います．白黒モードの場合もカラーモードの場合も，画面は１枚のみでその両方を同時に使用することはできません．

画面モード	SCREEN〔第１パラメータ〕
白黒モード	SCREEN 0
カラーモード	SCREEN 1

4.1.2　画面ポジション

N_{100}-BACIC は，ディスプレイ画面に対して，キャラクタやグラフィックスを横長方向に表示したり縦長方向に表示したりすることができます．この画面ポジションの設定は，SCREEN 文の第２パラメータで行います．

表示	横　方　向	縦　方　向
SCREEN ,〔第2パラメータ〕	SCREEN ,0	SCREEN ,1
横×縦（ピクセル）	720×512	512×720
ディスプレイ画面		

このように横表示，縦表示の選び方によって座標系も異なります．また，画面内に表示されるキャラクタの行数とケタ数も横表示か縦表示かによって，次のような組合せが可能です．これらの設定はWIDTH文によって行います．

横表示（桁×行）	縦表示（桁×行）
80×25	64×45
80×32	85×45
80×64	85×90
90×25	—
90×32	—
120×25	—
120×32	—
120×64	—

横表示の初期状態は80行×25桁
縦表示の初期状態は64行×45桁

どのような設定の状態であっても，画面ポジションを変えると，初期状態に設定されます．
ただし，WIDTH USING文を使用すれば，これ以外の表示方法を設定することも可能です．

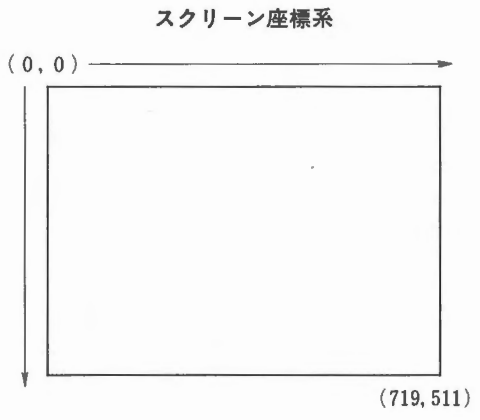
4.2 ウィンドウとビューポート

4.2.1 ワールド座標系とスクリーン座標系

ディスプレイ画面上の座標系には，スクリーン座標系とワールド座標系の2つの座標系があります．この2つの座標系は，グラフィックに対して有効です．

スクリーン座標系

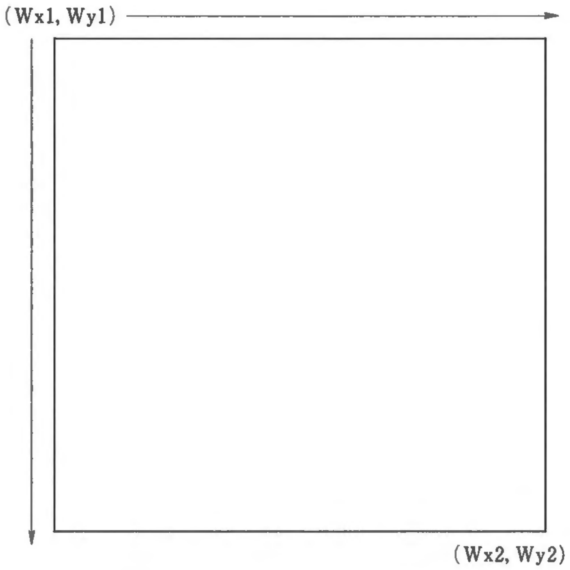
ビューポートを設定する際，用いられる座標系です．その物理的な大きさは，ディスプレイ画面と一致しており，座標上の各点が画面の1ピクセルと常に一致しています．つまり，左上の頂点を(0, 0)とし右下の頂点を（719, 511）とする座標系です．



ワールド座標系

ユーザーが使うことのできる最大の座標系で，BACIC が設定した仮空の座標です．

ワールド座標系



$|Wx2-Wx1| < 9.99999E+62$
$|Wy2-Wy1| < 9.99999E+62$

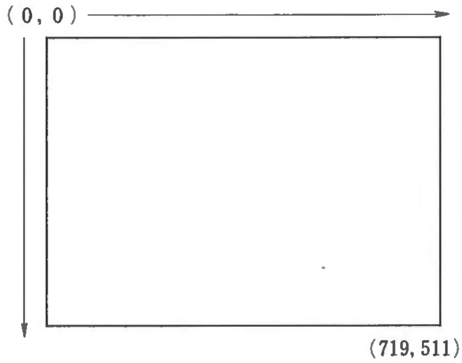
ワールド座標系の大きさは縦•横ともに－9.99999E＋62～＋9.99999E＋62の範囲まで許されます．この範囲内の領域なら，どこでも表現することができます．ただし，水平方向の幅（Wx2－Wx1）及び垂直方向の幅(Wy2－Wy1)が，9.99999E＋62を越えてはなりません．PSET(点を打つ)，LINE（線を引く）などのほとんどのグラフィックス命令が，ワールド座標に対して働きます．

4.2.2　ウィンドウ

ワールド座標とは，元々仮空のものであり，我々が実際に見ることのできる座標はスクリーン座標です．したがってユーザーは，ワールド座標上の任意の座標を指定して，スクリーン座標上に表示領域として割り当てなくてはなりません．そのための操作を行うのが，WINDOW 文です．つまり，WINDOW 文による指定により，ユーザーはスクリーンの物理的な大きさに制限されないスペースで，線を引いたり，グラフや物を描くことができます．この指定されたワールド座標上の領域をウィンドウと呼びます．
WINDOW 文には，SCREEN オプションの有無について2つの指定方法があり，その初期状態はスクリーン座標系（(0, 0) と (719, 511) を対角点とする四角形）と同じです．
引数を用いずに WINDOW 文を実行すると初期状態になりますが，その場合 WINDOW 変換は行いません．その場合の座標系は次の例のようになります．

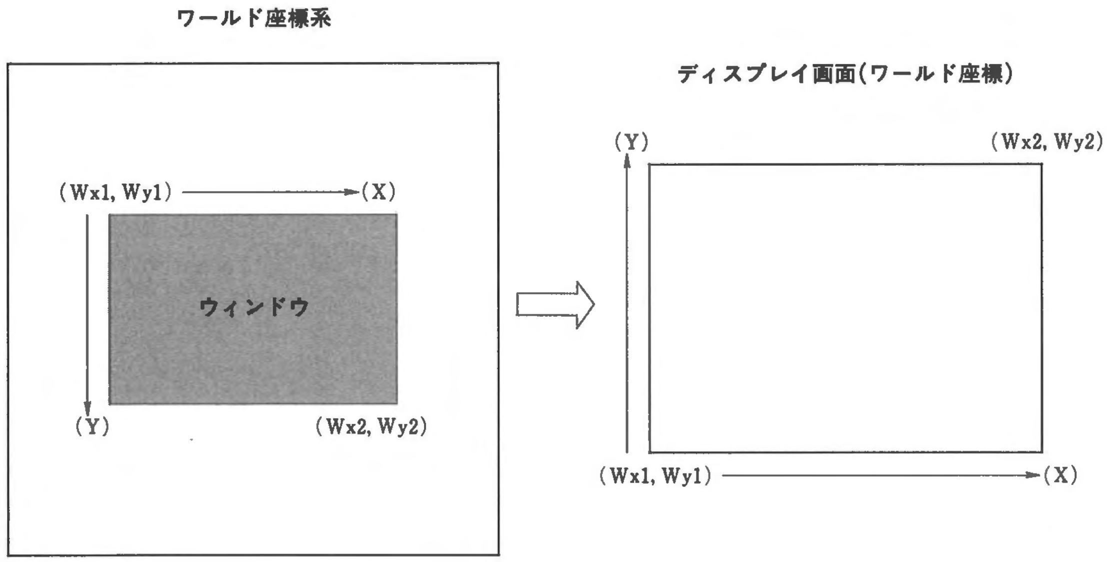
例）　WINDOW

ディスプレイ画面



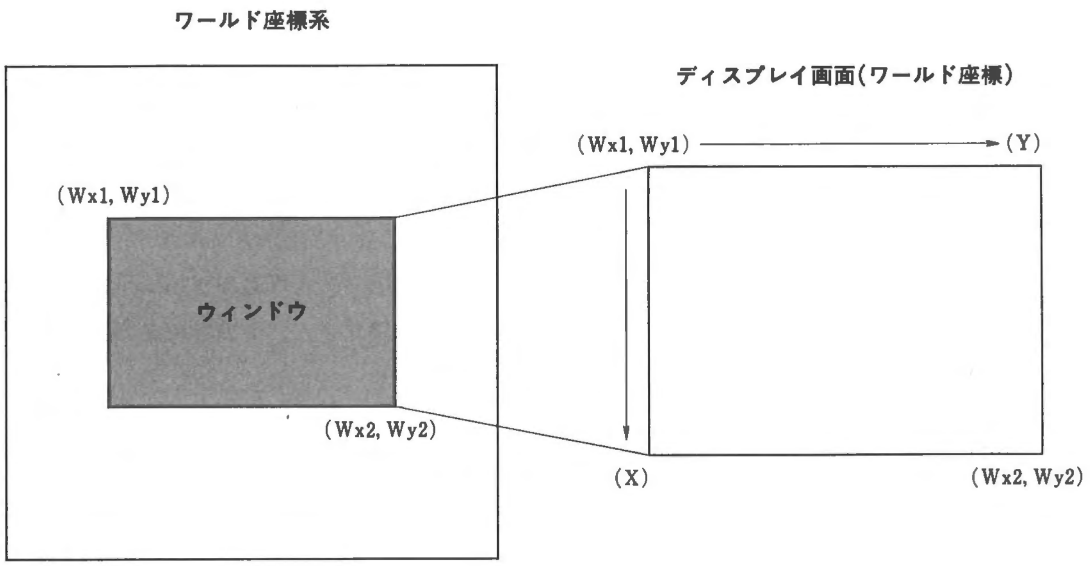
WINDOW 文の指定方法

① WINDOW (Wx1, Wy1)−(Wx2, Wy2)



WINDOW 指定を行うと，ワールド座標系の（Wx1, Wy1）から（Wx2,Wy2）の領域をウィンドウとしています．その際，ワールド座標のY軸が反転するので，ディスプレイ画面上に正しいデカルト座標を表示させることができます．

② WINDOW SCREEN (Wx1, Wy1)−(Wx2, Wy2)

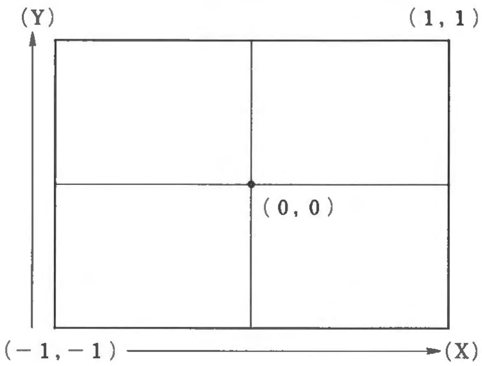


このSCREEN オプションつきの指定とWINDOW 指定の違う点は，Y軸が反転しないということです．

次に各々の場合の例を示します．

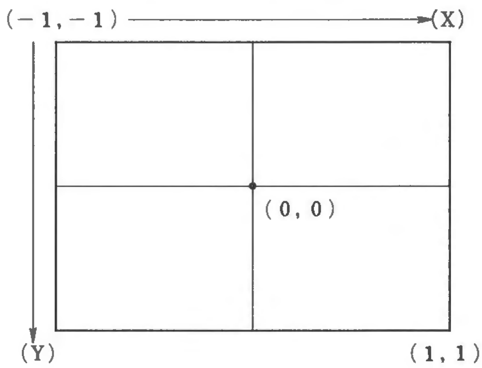
例1）　WINDOW（−1, −1）−（1, 1）

を実行すると，ディスプレイ画面は次のようになり，Y軸は反転します．



例2）　WINDOW SCREEN（−1, −1）−（1, 1）

を実行すると，ディスプレイ画面は次のようになり，Y軸は反転しません．



4.2.3　ビューポート

ウィンドウは，ワールド座標系においてユーザーがどの領域を表示領域とするかを設定するものでしたが，ビューポートは，WINDOW 文によって指定された領域をディスプレイ画面のどこに表示させるかの設定です．つまり，ワールド座標系の各ピクセルがビューポートという物理的な座標系に写像されるわけです．

この操作を VIEW 文によって行いますが，その指定方法には SCREEN オプションの有無により次の2通りがあります．また，初期状態はスクリーン座標系と同じです．

① VIEW（Sx1, Sy1）−（Sx2, Sy2）

0≦Sx1, Sx2≦719

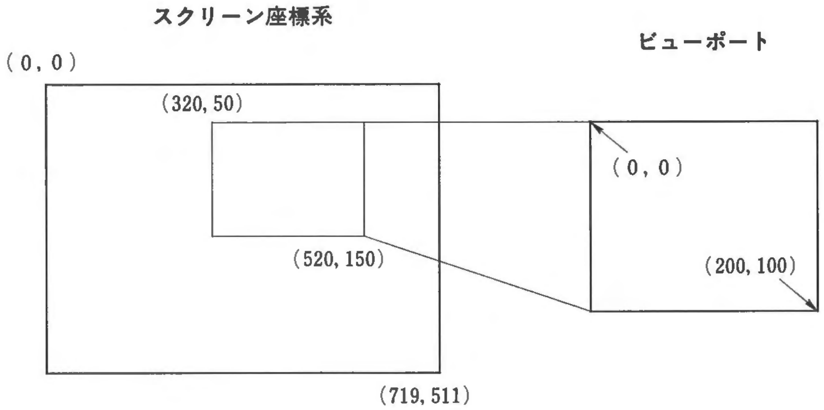
0≦Sy1, Sy2≦511

この場合，スクリーン座標系の（Sx1, Sy1）および（Sx2, Sy2）を対角点とする四角形がビューポートとなります．ビューポート内の座標系は，WINDOW 文に引数がある場合とない場合では異なります．

● WINDOW 文の引数がない場合（初期状態）

ビューポート内の座標はスクリーン座標系で，(Sx1, Sy1) を基点とする相対座標です．

例) WINDOW
　　VIEW (320, 50) − (520, 150)



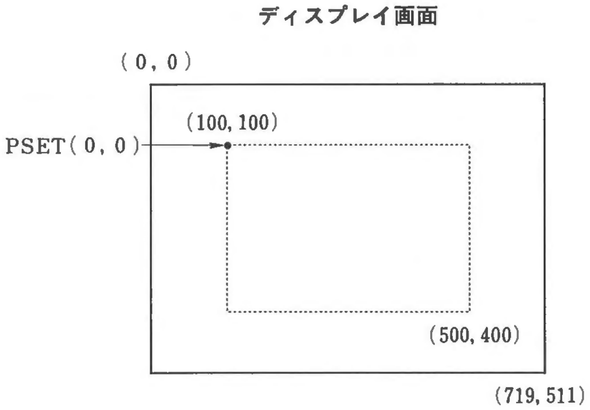
また，次のような命令

VIEW (100, 100) − (500, 400)

に対して，

PSET (0, 0)

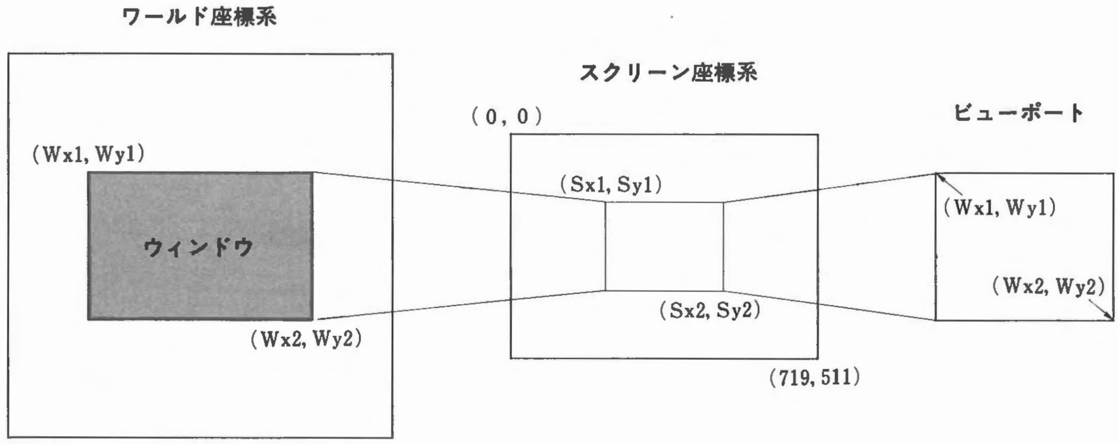
を指定すると，この点はディスプレイ画面上の (100, 100) の位置に現われます．



● WINDOW SCREEN (Wx1, Wy1) − (Wx2, Wy2)

ビューポート内の座標系はワールド座標となり，ビューポートにおける最も一般的な形です．

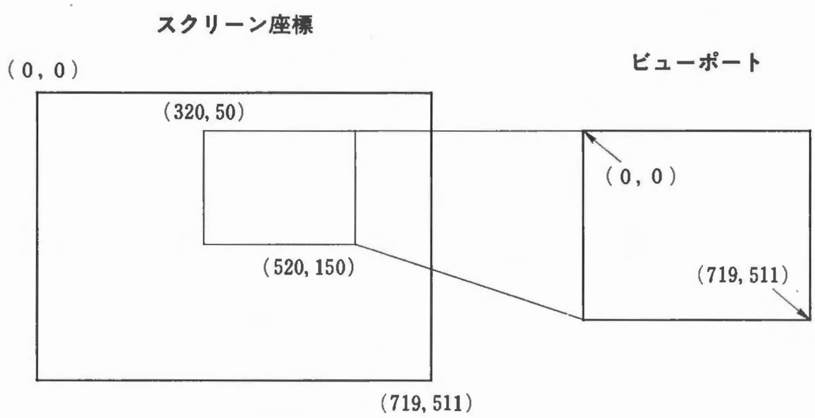
例) WINDOW SCREEN (Wx1, Wy1) − (Wx2, Wy2)
　　VIEW (Sx1, Sy1) − (Sx2, Sy2)



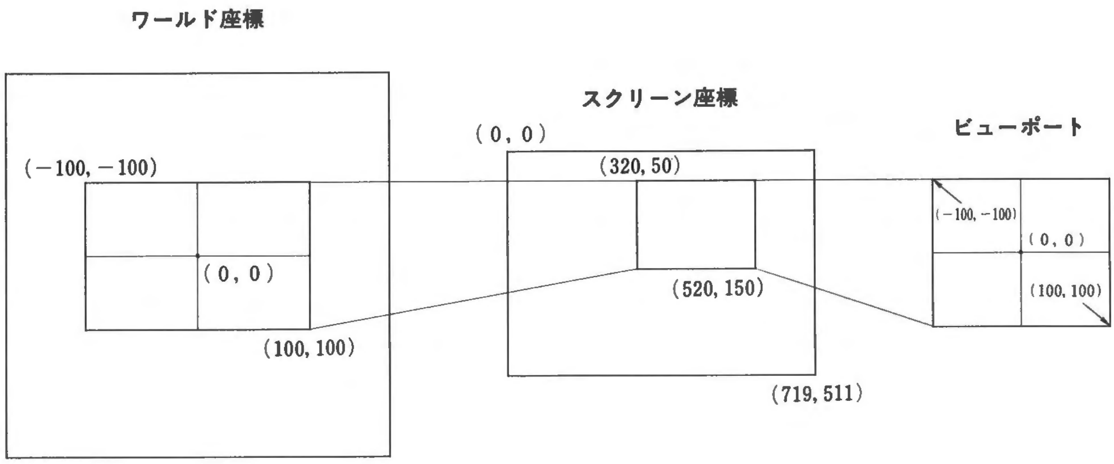
次に示す例 1 で，ウィンドウがスクリーン座標系と一致する特別な場合を挙げ，例 2 では，一般的な場合を挙げます．

また，WINDOW (Wx1, Wy1)－(Wx2, Wy2) で設定するとビューポートもデカルト座標になります．

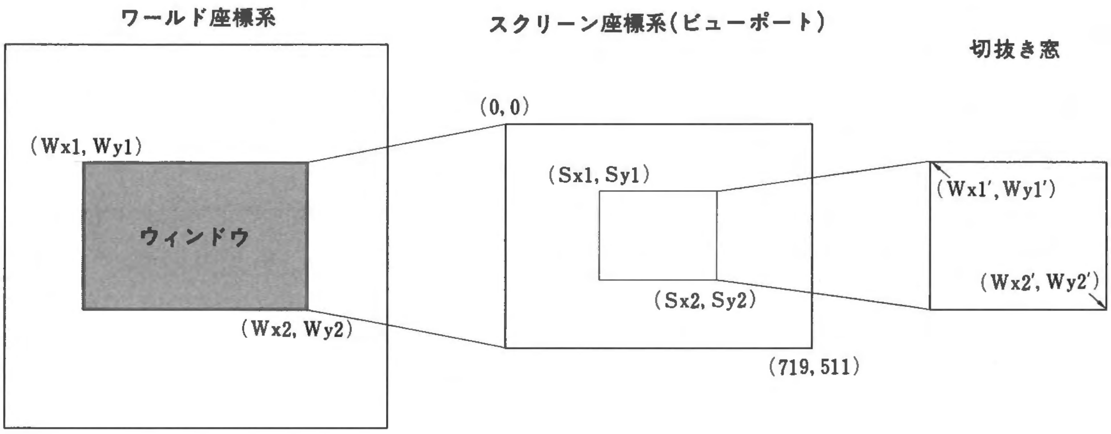
例 1)　WINDOW SCREEN (0, 0)－(719, 511)
　　　VIEW (320, 50)－(520, 150)



例2)　WINDOW SCREEN (−100, −100)−(100, 100)
VIEW (320, 50)−(520, 150)

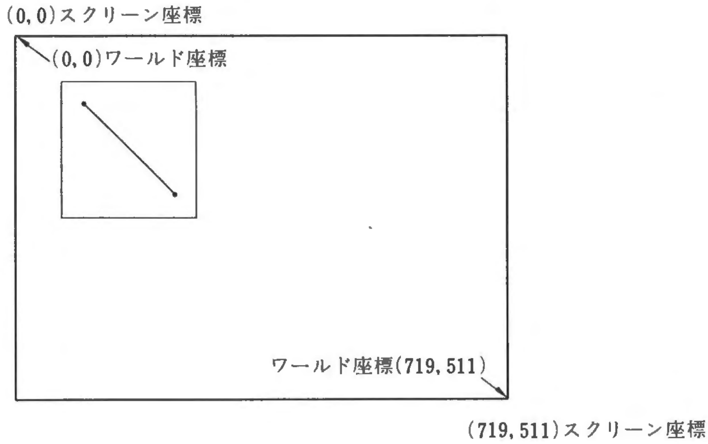


② VIEW SCREEN (Sx1, Sy1)−(Sx2, Sy2)

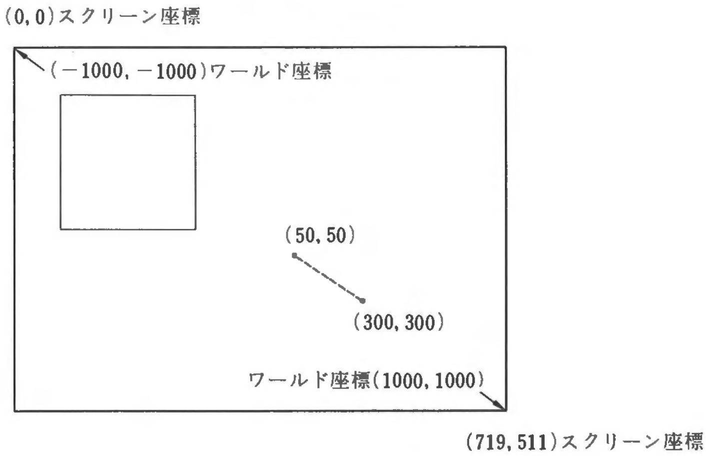


この場合，スクリーン座標系の (Sx1, Sy1) と (Sx2, Sy2) を対角点とする四角形は，前で述べたビューポートとは異なり単なる〝切抜き窓〟の働きをします．SCREEN オプションを付加すると，WINDOW 文で指定した領域（ウィンドウ）は常にディスプレイ画面の全体 (0, 0) と (719, 511) を対角点とする四角形）に写像されます．つまり，今までのビューポートの考え方でいけば，ディスプレイ画面の領域がビューポートの役目をするわけです．そしてこの〝切抜き窓〟を指定することによって，そのウィンド内のワールド座標系の表示領域が制限されることになるわけです．わかりやすくするために，実際に例を用いて説明します．次の例1と例2を比べてみてください．

例 1）　WINDOW
　　　VIEW SCREEN (50, 50)－(300, 300),, 15
　　　LINE (100, 100)－ (250, 250)



例 2）　WINDOW SCREEN (－1000, －1000)－ (1000, 1000)
　　　VIEW SCREEN (50, 50)－ (300, 300),, 15
　　　LINE (100, 100)－ (250, 250)



例 2 では画面上に直線が現われません。実はこの直線は図の点線のところに位置しているのです。つまり，この〝切抜き窓″だけが表示可能な領域となるわけです。
また，次のような命令

　VIEW SCREEN (100, 100)－(500, 400)　(WINDOW は初期状態)

に対して

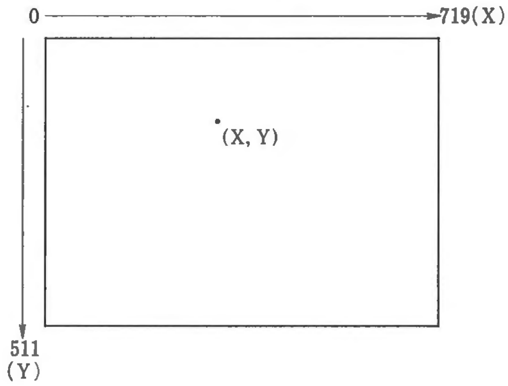
4.3 座標の指定の仕方

2つの座標，絶対座標と相対座標について説明します．絶対座標は最も一般的な座標の指定です．また点を打つ(PSET)，線を引く(LINE)などに使う座標の指定には，相対座標による方法も使うことができます．

4.3.1 絶対座標による指定の仕方

絶対座標とは，X座標，Y座標が決まることによって一意的に決まる座標のことです．下図のように常に左上の頂点が（0，0）で，右下の頂点は（719，511）となります．絶対座標による指定とは，この座標系を使って（X，Y）の位置を表わす方法をいいます．

絶対座標



絶対座標による指定は．すべて（X，Y）という形になって表われます．

4.3.2 相対座標による指定の仕方

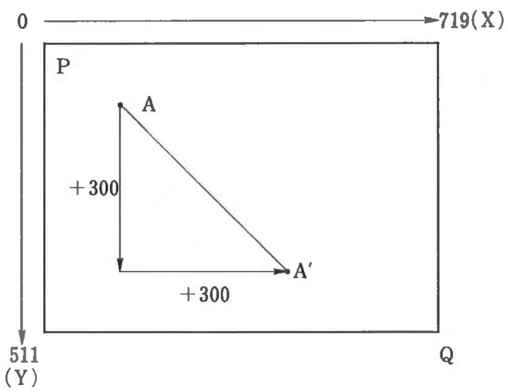
相対座標とは，座標系内にある1点を基点とした時，X方向，Y方向にどれだけ相対的に移動するかによって決められる座標です．相対座標による指定の仕方とは，この相対座標を使って位置を表わす方法をいいます．

例えば，次のような座標系があり，基点をA（100，100)とします．

この場合左上の頂点Pは（−100，−100)，右下の頂点Qは（+619，+411）となります．

	P	Q	A
絶対座標	(0，0)	(719，511)	(100，100)
相対座標	(−100,−100)	(+619,+411)	(0，0)

相対座標



前図で，絶対座標ではA′は(400，400)ですが，Aを基点とした相対座標で表わすとA′は(+300，+300)となります．また，A′を基点とすると，新しい相対座標が作られ，Aは(−300，−300)，Pは(−400，−400)となります．

この指定方法の優れている点は，相対移動量(x，y)を常に同じ値であっても，基点の位置を動かすことによって違った位置を表現できるということです．N_{100}-BASICのグラフィック命令で相対座標を使って座標指定を行う場合は

STEP (x, y)

を使います．ここで(x，y)は相対量を表わします．N_{100}-BASICのグラフィック命令で相対座標の指定が許されているのは，

PSET	PRESET
LINE	PAINT
CIRCLE	GET@(但し終点座標のみ)

の6つです．

例えば，三角形を絶対座標と相対座標を使って描いてみると，

絶対座標		相対座標	
	10 PSET (100, 100)		10 PSET (100, 100)
	20 LINE—(50, 400)		20 LINE—STEP (−50, 300)
	30 LINE—(300, 400)		30 LINE—STEP (250, 0)

となり，どちらも同じ三角形を描きます．

プログラムリスト

```
100 CLS:PRINT TIME$:KK=5/6:LL=100:RR=15:RL=15
110 DIM SS(20),I(20)
130 ANGLE=90:N=0:PP=ATN(1)*8/360
140 LINE(360,510)-STEP(0,-LL)
150 SS(N)=LL*KK
160 SIZE=SS(N)
170 IF SIZE<20 THEN 270
180 AD=2*RR+RL:GOSUB 300
190 I(N)=0
200   AD=-RR-RL:GOSUB 300
210   LINE-STEP(-DX*SIZE,-DY*SIZE)
220   N=N+1:SS(N)=SIZE*KK:GOTO 160
230   N=N-1:SIZE=SS(N)
240   LINE-STEP(DX*SIZE,DY*SIZE)
250 I(N)=I(N)+1:IF I(N)<2 THEN 200
260 AD=RL:GOSUB 300
270 IF N<>0 THEN 230 ELSE PRINT TIME$:END
300 ANGLE=ANGLE+AD
310 DX=COS(ANGLE*PP):DY=SIN(ANGLE*PP)
320 RETURN
```

4.4　カラーパレット

PALETTE文を使って画面の色，画面上の文字や図形の色を指定したり，一度指定した色を変えることができます．

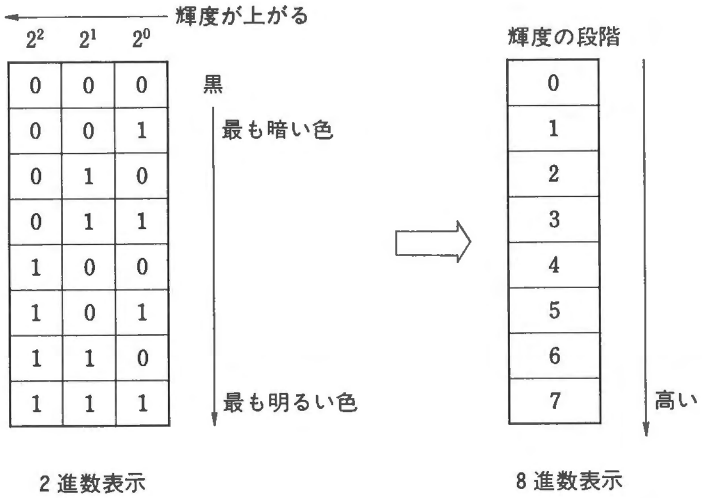
4.4.1　パレット番号

画面に表示される図形の最小単位をピクセル（画素）といいます．1ピクセルは4ビットから構成され，その4ビットの組合せ($2^4=16$通り)によって色を指定する番号が決まります．この16種類の番号をパレット番号といいます．N_{100}-BASICでの色の指定は，これらのパレット番号をもつパレットで行われます．

4.4.2　カラーコード

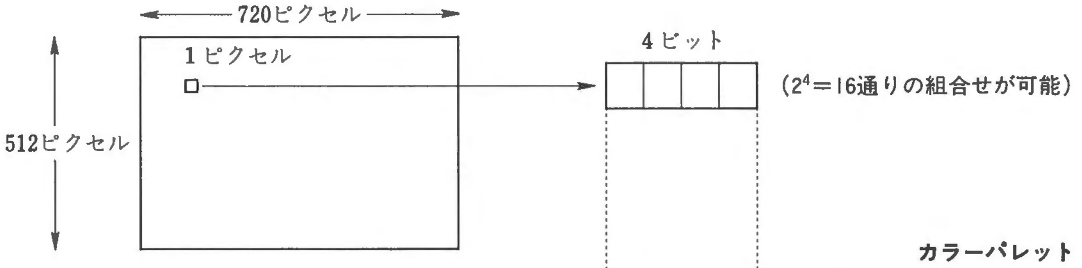
各パレットは，R（赤），G（緑），B（青）の3色のカラーコードをもっています．各色3ビットから構成され，この3ビットの組合せ（$2^3=8$通り）で色の輝度を表わしています．

例）　B（青）のカラーコード



このように1色につき8段階の輝度を表現することが可能なので，8進数を用いると便利です．従ってカラーコード全体としては，$8^3=512$種類の色を表現することができます．

パレット番号とカラーコードの関係は次のようになります。



カラーパレット

パレット番号	2進表示	B			G			R		
		2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
0	0 0 0 0									
1	0 0 0 1									
⋮	⋮									
⋮	⋮									
14	1 1 1 0									
15	1 1 1 1									

このように1ピクセルをぬりつぶす場合，そのピクセルがもっている4ビットでパレット番号を表わします。16個のパレット番号にどのカラーコードを割り付けるかは，PALETTE文を使って次のように設定します。

例）　パレット番号1に最も明るい青を設定する場合

パレット番号		カラーコード			
10進	2進	B	G	R	8進数表示
1	0 0 0 1	1 1 1	0 0 0	0 0 0	7 0 0

```
PALETTE 1,&O700
```

初期状態のパレット番号とそれに対応する色の関係は次のようになります。

パレット番号	色	B 2^2	B 2^1	B 2^0	G 2^2	G 2^1	G 2^0	R 2^2	R 2^1	R 2^0	8進数表示
0	黒	0	0	0	0	0	0	0	0	0	0 0 0
1	暗い青	1	0	1	0	0	0	0	0	0	5 0 0
2	暗い緑	0	0	0	1	0	1	0	0	0	0 5 0
3	暗いシアン	1	0	1	1	0	1	0	0	0	5 5 0
4	暗い赤	0	0	0	0	0	0	1	0	1	0 0 5
5	暗いマゼンタ	1	0	1	0	0	0	1	0	1	5 0 5
6	暗い黄	0	0	0	1	0	1	1	0	1	0 5 5
7	暗い白	1	0	1	1	0	1	1	0	1	5 5 5
8	灰	0	1	1	0	1	1	0	1	1	3 3 3
9	青	1	1	1	0	0	0	0	0	0	7 0 0
10	緑	0	0	0	1	1	1	0	0	0	0 7 0
11	シアン	1	1	1	1	1	1	0	0	0	7 7 0
12	赤	0	0	0	0	0	0	1	1	1	0 0 7
13	マゼンタ	1	1	1	0	0	0	1	1	1	7 0 7
14	黄	0	0	0	1	1	1	1	1	1	0 7 7
15	白	1	1	1	1	1	1	1	1	1	7 7 7

プログラムリスト

```
100 SCREEN 1,0,0,1
110 CLS
120 LOCATE 1,1,0
130 FOR J=0 TO 1
140 FOR I=0 TO 7
150 LINE(72*I,100*J)-STEP(72,72),J*8+I,BF
160 NEXT I
170 NEXT J
180 FOR J=0 TO 1
190 LOCATE 5*J+5
200 FOR I=0 TO 7
210 PRINT USING "    ##  ";J*8+I;
220 NEXT I
230 PRINT
240 NEXT J
250 LOCATE 20,20:PRINT "PALETTE  ";:PRINT USING "##";
P;:PRINT ", &O";:PRINT USING "##,";R;G;B
260 LOCATE 21,10:INPUT "パレット ナンバー は??",P
270 IF P<0 OR P>15 THEN 260
280 LOCATE 22,10:INPUT "ブルー の輝度は?",B
290 IF B<0 OR B>7 THEN LOCATE 22,28:PRINT "      ":GOTO 280
300 LOCATE 22,10:INPUT "グリーンの輝度は?",G
310 IF G<0 OR G>7 THEN LOCATE 22,28:PRINT "      ":GOTO 300
320 LOCATE 22,10:INPUT "レッド の輝度は?",R
330 IF R<0 OR R>7 THEN LOCATE 22,28:PRINT "      ":GOTO 320
340 CO=(B*8+G)*8+R
350 PALETTE P,CO
360 GOTO 250
```

複数のパレット番号に1つのカラーコードを対応させることはできますが，その逆はできません．

例）
```
100 '***** PALETTE *****
110 SCREEN 1,0:CLS
120 FOR I=1 TO 50
130 X=RND*719
140 Y=RND*511
150 R=RND*80
160 C=INT(RND*7+1)
170 CIRCLE(X,Y),R,C
180 PAINT(X,Y),C
190 NEXT I
```

このプログラムの実行後に，

PALETTE 1, &O005

を実行すると，パレット番号1の色は瞬時に赤に変わります．

4.5　ラインスタイルとタイリング

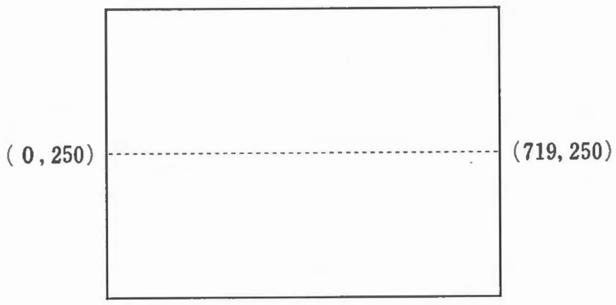
4.5.1　ラインスタイル

ラインスタイルとはLINE文にある機能で，引く線の型態を設定します．これを設定するためには，16ビットで表現可能な数（&H0～&HFFFF）の範囲内の値を用います．ラインスタイルは16進数で指定しなくてもかまいませんが，16進数で表現すると便利です（詳細は，N_{100}-BASICリファレンスマニュアルのLINE文の項を参照）．

例えば，

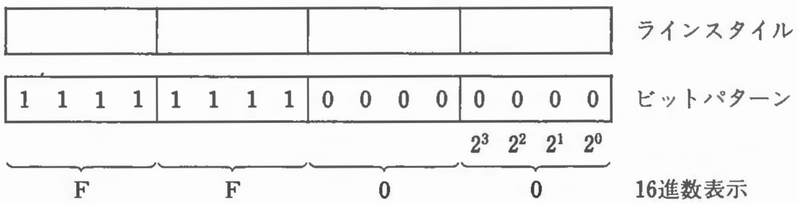
LINE (0, 250)－(719, 250), 3, , &HFF00

を実行すると，下図のような点線が引かれます．



ラインスタイルを&HFF00と設定すると次のようになります．

ラインスタイル（&HFF00）



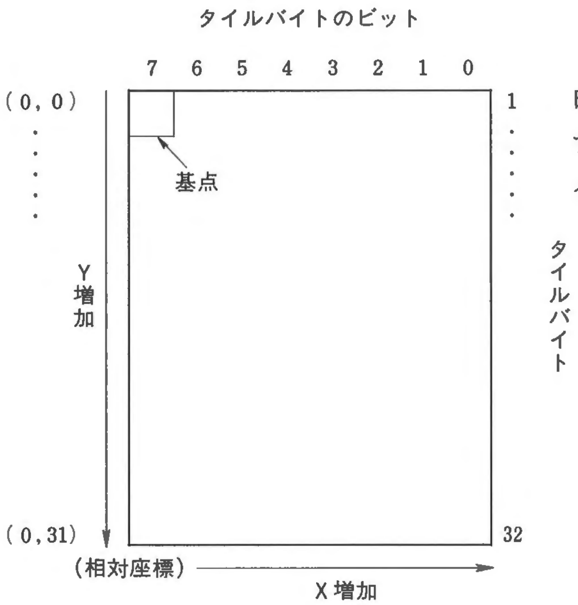
つまり，このラインスタイルが（0，250）～(719，250）まで繰り返されるため，図のような点線が現われます．

4.5.2 タイリング

タイリングとはPAINT文にある機能で，タイルストリングによって設定された模様（タイル）で，指定領域内を埋めるときに使います（タイルストリングに関する詳細はN_{100}-BASICリファレンスマニュアルのPAINT文の項を参照）．

タイルの大きさは，幅（横方向）は常に8ビットとなり，長さ（縦方向）は白黒モードの場合も，カラーコードの場合も1～64バイトとなります．

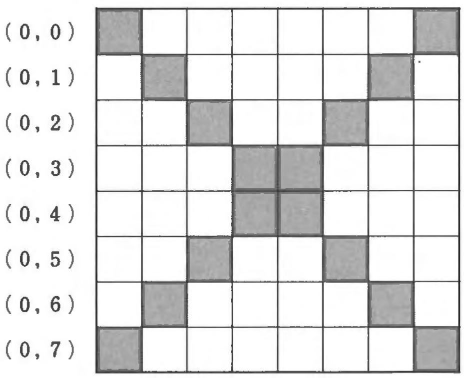
白黒モードの場合



白黒モードの場合は，横8ピクセルを1バイトと扱います．8ピクセルの表現の仕方は，前に述べたラインスタイルの考え方と同様です．

次に挙げる例は，画面上を〝X″模様でぬりつぶします．

例） PAINT (320, 100), CHR$(&H81)+CHR$(&H42)+CHR$(&H24)+CHR$(&H18)+CHR$(&H18)+CHR$(&H24)+CHR$(&H42)CHR$(&H81)



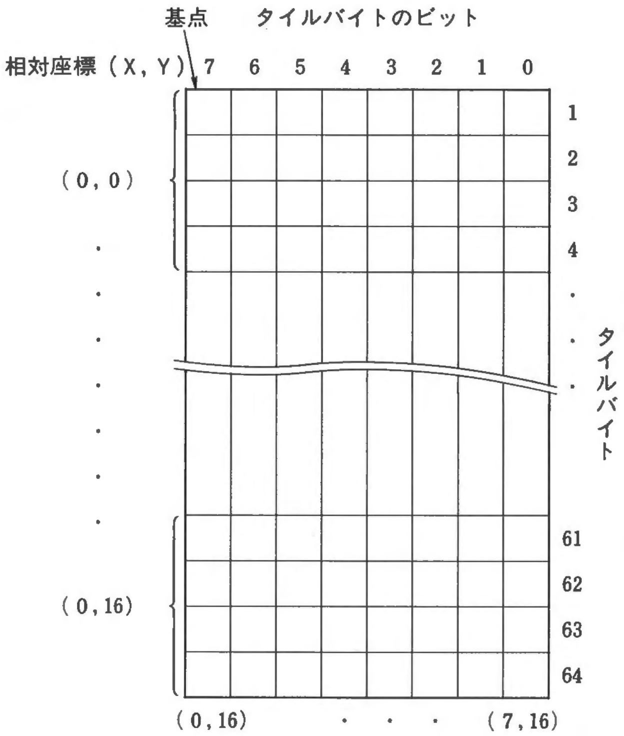
CHR$(&H81)	タイルバイト1
CHR$(&H42)	タイルバイト2
CHR$(&H24)	タイルバイト3
CHR$(&H18)	タイルバイト4
CHR$(&H18)	タイルバイト5
CHR$(&H24)	タイルバイト6
CHR$(&H42)	タイルバイト7
CHR$(&H81)	タイルバイト8

またこのとき，

PAINT (320, 100), CHR$ (&H81, &H42, &H24, &H18, &H18, &H24, &H42, &H81)

と表現することもできます．

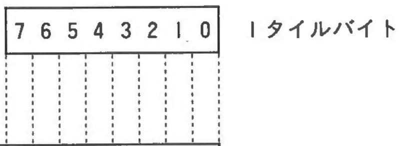
カラーモードの場合



カラーモードの場合は，4バイトで横8ピクセルを表わします．つまり，縦方向にnバイトのタイルを指定するには4＊nバイト（1≦n≦16）必要となります．各ピクセルにおける色の指定方法は，カラーパレットの考え方と同様ですが，ここでは横8ピクセルの色指定を同時に行います．

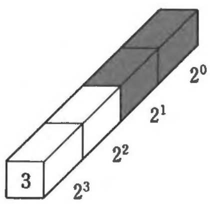
例えば，1バイトのタイルを考えてみましょう．1バイトには8ピクセルあるので，各々のピクセルに下図のようなパレット番号の色でぬりつぶすことにします．

タイルバイトのビット



ピクセル番号	1	2	3	4	5	6	7	8
パレット番号	3	3	3	4	5	6	7	8

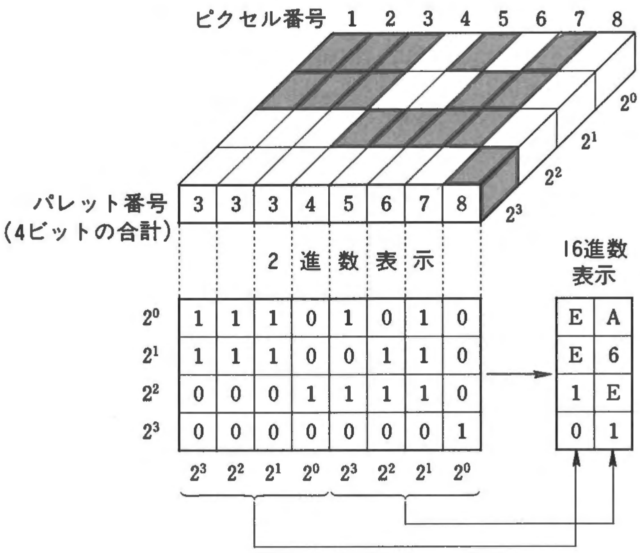
1ピクセルは4ビットもっているので，その4ビットを使って指定したい色を表現します．つまり，1番のピクセルについて考えると，このピクセルをパレット番号3の色でぬりつぶすには，3の値をこの4ビットで表現するということです．したがって下図のようになります．



$2^0\times1+2^1\times1+2^2\times0+2^3+0=3$

1番目のピクセル

8つのピクセル全体については次のようになります.



したがって, 8ピクセルの色を決めるタイルストリングは次のようになります.

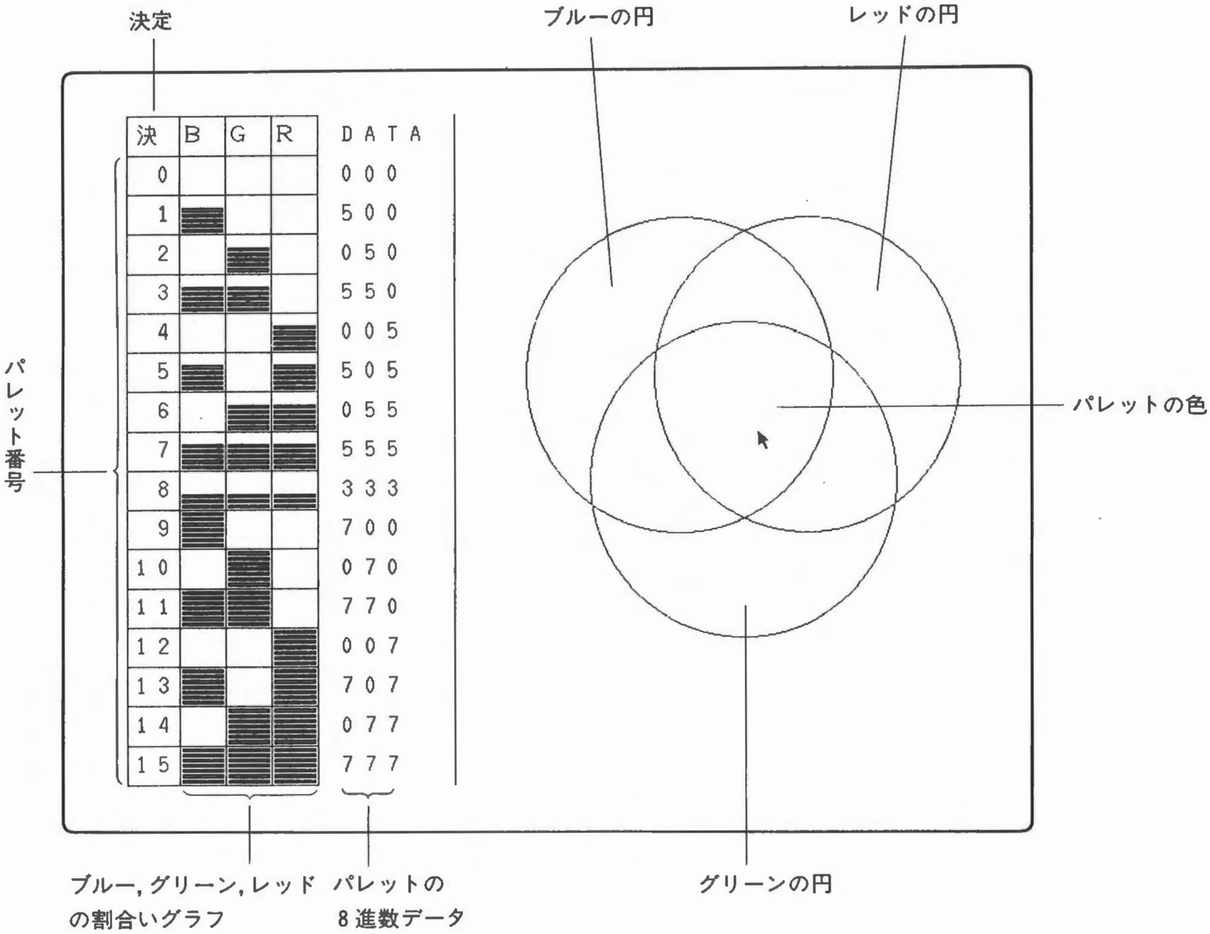
CHR$ (&HEA)+CHR$ (&HE6)+CHR$ (&H1E)+CHR$ (&H01)

4.6　応用プログラム

4.6.1　パレット応用プログラム

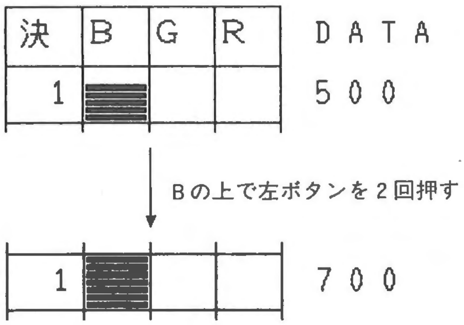
PALETTE文を使った応用プログラムです．B(青)，G(緑)，R(赤)の輝度を変えて，いろいろな中間色を出すことができます．このプログラムは512色をすべて簡単に表示することができ，またそのデータも表示してくれます．

1．プログラムを実行すると，次のような画面が表示されます．

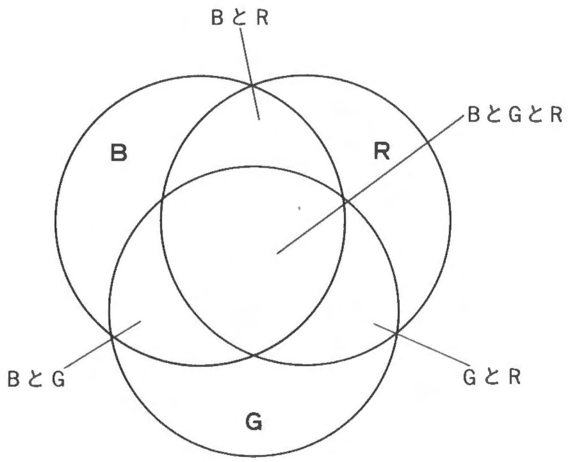


2．カーソルを，変更したいパレット番号（0～15）の上に動かして左ボタンを押してください．すると，そのパレットの現在のB・G・Rの状態が，右の円に表示されます．

3．3原色（青，緑，赤）のうち，輝度を変更したい色の上にカーソルを移動します．輝度を増やしたい場合は左のボタンを，減らしたい場合は右のボタンを押してください．それぞれの動作に応じて，選択されているパレット番号のグラフ，及びDATAが変化します．輝度は8進数（0～7）の8段階で表示されます．0が最も暗く，数字が増えるごとに明るくなり，7で最も明るくなります．



3つの円が重なっている場所の色が，そのパレットの色となります．また，図のように円が重なっている場所でボタン操作を行なうと，重なっている3原色がすべて変化します．例えば，中心でボタン操作を行った場合，BとGとRが同時に変化します．



4．変更したパレットの色を見たい場合は，カーソルを決（決定）の上に移動させ，左ボタンを押してください．すると，円が消えてその場所に16色すべてが表示されます．表示されている8進数DATAに対応して，右横にそのデータから成る色を表示します．気に入った色ができた場合は，そのデータを書き留めておきましょう．

もとの画面に戻り，さらにパレットを変更したい場合は左のボタンを，終了したい場合は右のボタンを押してください．

5．データの使用法を説明します．

使いたいデータが〝514〞で，変更したいパレット番号が3の場合，

```
10 PALETTE 3, &O514
```

としてください．

6．ボタン操作のまとめ

パレット番号の選択
輝度の変更
決（16色表示）の実行
} 左ボタン

プログラムの終了――右ボタン

プログラムリスト

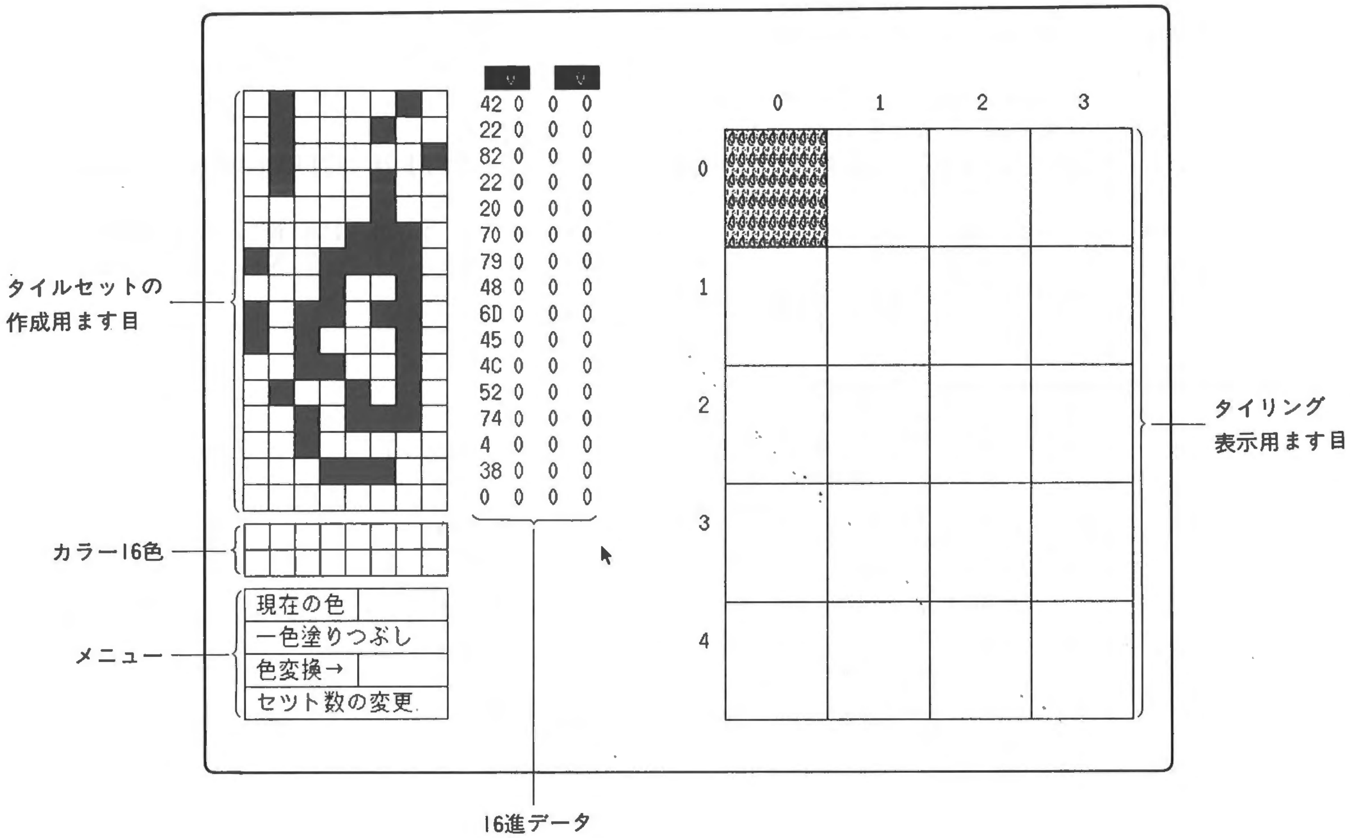
```
100 '****************************************
110 '***                                  ***
120 '***       PALETTE プログラム         ***
130 '***                                  ***
140 '****************************************
150 SCREEN 0,0,0,1:COLOR 1,0:CLS:LOCATE 5,1:PRINT"このプログ
ラムの使い方は、N100-BASICユーザーズガイドに記載されて
います。":FOR I=1 TO 1000:NEXT
160 IF INP(&H22) AND 2 THEN BEEP:BEEP:BEEP:PRINT:PRINT:PRINT
"                    カラーディスプレイ及びカラーボードを使用し
てください。":BEEP:PRINT:PRINT"                    プログラムを
終了させていただきます。":END
170 MOUSE 0:DIM PN(15,2),PD(15):SCREEN 1,0,0,0:PALETTE
180 DATA 0,0,0,5,0,0,0,5,0,5,5,0,0,0,5,5,0,5,0,5,5,5,5,5
190 DATA 3,3,3,7,0,0,0,7,0,7,7,0,0,0,7,7,0,7,0,7,7,7,7,7
200 FOR I=0 TO 15:A=64
210 FOR J=0 TO 2
220 READ PN(I,J):PD(I)=PD(I)+PN(I,J)*A
230 A=A/8:NEXT:NEXT
240 WIDTH USING 0,18,30,0,711
250 LOCATE 1,1,0
260 FOR I=2 TO 17
270 IF I<12 THEN LOCATE I,2:PRINT I-2:GOTO 290
280 LOCATE I,1:PRINT I-2;
290 NEXT
300 LOCATE 1,2:PRINT "決":LOCATE 1,4:PRINT"B":LOCATE 1,6:PR
INT"G":LOCATE 1,8:PRINT"R":LOCATE 1,11:PRINT"DATA"
310 LINE(10,0)-(160,0)
320 FOR Y=31 TO 511 STEP 30
330 LINE(10,Y)-(160,Y)
340 NEXT
350 LINE(10,0)-(10,511)
360 FOR X=52 TO 160 STEP 36
370 LINE(X,0)-(X,511)
380 NEXT
390 LINE(268,0)-(268,511)
400 FOR I=0 TO 15:X=54:FOR J=0 TO 2
410 FOR H=59+30*I TO 59+30*I-4*(PN(I,J)-1) STEP -4
420 IF PN(I,J)=0 THEN 440
430 LINE(X,H)-STEP(32,-2),15,BF
440 NEXT:X=X+36:NEXT:NEXT
450 FOR Y=2 TO 17
460 LOCATE Y,11:PRINT RIGHT$("00"+OCT$(PD(Y-2)),3);
470 NEXT
480 GOSUB 1120
490 MOUSE 1,0,0,1
500 '***** MAIN *****
510 X=MOUSE(0):Y=MOUSE(1)
520 IF X<52 THEN 560
530 IF X>270 THEN 660
540 GOTO 510
```

```
550 '***** 選択 *****
560 IF MOUSE(2,1)=0 THEN 510
570 IF Y<31 THEN GOSUB 1180
580 CPN=INT((Y-32)/30)
590 B=PN(CPN,0):G=PN(CPN,1):R=PN(CPN,2)
600 YB=59+30*CPN-4*B:XB=54
610 YG=59+30*CPN-4*G:XG=90
620 YR=59+30*CPN-4*R:XR=126
630 GOSUB 1020
640 GOTO 510
650 '***** C I R C L E *****
660 IF MOUSE(2,1)=MOUSE(2,2) THEN 510
670 C=POINT(X-2,Y-2):IF C=0 OR C=15 THEN 510
680 IF MOUSE(2,1)=1 THEN AD=1
690 IF MOUSE(2,2)=1 THEN AD=-1
700 ON C GOSUB 730,800,870,890,960,980,1000
710 GOTO 510
720 'BLUE
730 B=B+AD
740 IF AD=1 THEN IF B>7 THEN B=7 ELSE LINE(XB,YB)-STEP(32,-2
),15,BF:PD(CPN)=PD(CPN)+64:YB=YB-4
750 IF AD=-1 THEN IF B<0 THEN B=0 ELSE LINE(XB,YB+1)-STEP(32
,3),0,BF:PD(CPN)=PD(CPN)-64:YB=YB+4
760 PN(CPN,0)=B:LOCATE CPN+2,11:PRINT RIGHT$("00"+OCT$(PD(CP
N)),3);
770 GOSUB 1020
780 RETURN
790 'GREEN
800 G=G+AD
810 IF AD=1 THEN IF G>7 THEN G=7 ELSE LINE(XG,YG)-STEP(32,-2
),15,BF:PD(CPN)=PD(CPN)+8:YG=YG-4
820 IF AD=-1 THEN IF G<0 THEN G=0 ELSE LINE(XG,YG+1)-STEP(32
,3),0,BF:PD(CPN)=PD(CPN)-8:YG=YG+4
830 PN(CPN,1)=G:LOCATE CPN+2,11:PRINT RIGHT$("00"+OCT$(PD(CP
N)),3);
840 GOSUB 1020
850 RETURN
860 'SIAN
870 GOSUB 730:GOSUB 800:RETURN
880 'RED
890 R=R+AD
900 IF AD=1 THEN IF R>7 THEN R=7 ELSE LINE(XR,YR)-STEP(32,-2
),15,BF:PD(CPN)=PD(CPN)+1:YR=YR-4
910 IF AD=-1 THEN IF R<0 THEN R=0 ELSE LINE(XR,YR+1)-STEP(32
,3),0,BF:PD(CPN)=PD(CPN)-1:YR=YR+4
920 PN(CPN,2)=R:LOCATE CPN+2,11:PRINT RIGHT$("00"+OCT$(PD(CP
N)),3);
930 GOSUB 1020
940 RETURN
950 'MAZENDA
960 GOSUB 730:GOSUB 890:RETURN
970 'KIIRO
980 GOSUB 800:GOSUB 890:RETURN
990 'WHITE
1000 GOSUB 730:GOSUB 800:GOSUB 890:RETURN
```

```
1010 '***** PALETTE SUB *****
1020 PB=B*64:PG=G*8:PR=R
1030 PALETTE 1,PB
1040 PALETTE 2,PG
1050 PALETTE 3,PB+PG
1060 PALETTE 4,PR
1070 PALETTE 5,PB+PR
1080 PALETTE 6,PG+PR
1090 PALETTE 7,PB+PG+PR
1100 RETURN
1110 '***** 円描き *****
1120 PALETTE:LINE(269,0)-(719,511),0,BF
1130 CIRCLE(495,280),120:CIRCLE(445,200),120:CIRCLE(545,200)
,120
1140 PAINT(400,130),1,15:PAINT(500,130),5,15:PAINT(600,130),
4,15:PAINT(400,250),3,15:PAINT(500,250),7,15:PAINT(580,250),
6,15:PAINT(500,340),2,15
1150 FOR I=0 TO 7:PALETTE I,0:NEXT
1160 RETURN
1170 '***** 決定 *****
1180 MOUSE 1,,,0:LINE(269,0)-(719,511),0,BF
1190 FOR I=0 TO 15:PALETTE I,PD(I):NEXT:I=0
1200 FOR Q=0 TO 4000:NEXT
1210 FOR Y=31 TO 481 STEP 30
1220 LINE(270,Y)-(711,Y+30),I,BF
1230 I=I+1:NEXT
1240 IF MOUSE(2,2)=1 THEN PALETTE:SCREEN 1,0,0,0:WIDTH 80,25
:CLS:END
1250 IF MOUSE(2,1)=0 THEN 1240
1260 GOSUB 1120:MOUSE 1,0,0,1:RETURN
```

4.6.2 タイリング応用プログラム

PAINT文のタイリングの機能を使ったプログラムです。マウスをフルに活用しているので，タイルパターンの作成がとても楽にできます。また，データの表示も自動的に行ってくれるので，めんどうな計算もしなくてすみます。ぜひ一度使ってみてください。



プログラムを実行すると，セット数をたずねてくるので1～16の数字を入力してください。すると，図のような画面が表示されます。

1. **タイルセットの作り方**

   カーソルをメニュー最上部に移動し，左ボタンで色を選びます。そして，作成用ます目に移動させ，塗りたい場所の上で左ボタンを押します。

2. **タイリングの実行方法**

   右ボタンを押します。すると作成用ます目の右がわに，作成されたパターンの16進データが表示されてゆき，それが終わると右のます目に作成したタイルセットでペイントします。

3. **メニューの説明**（コマンドおよび色の選択は全て左ボタンです）

- 現在の色……………現在選んでいる色を表示します。
- 一色塗りつぶし……作成用ます目を，〝現在の色〟で塗りつぶします。
- 色変更………………ます目にある〝現在の色〟を，指定した色に変えます。右ボタンを押すと実行されます。
- セット数の変更……タイルセットの数を変更したいときに選びます。

4.　データの見かた

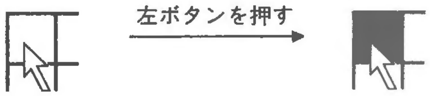
表示されるデータは"&H"を省略した16進数です．データの上の数字は，表示用ます目の番号であり，これによって現在どのパターンのデータが表示されているかがわかります．気に入ったパターンのデータは，きちんと記録しておきましょう．

5.　特殊操作

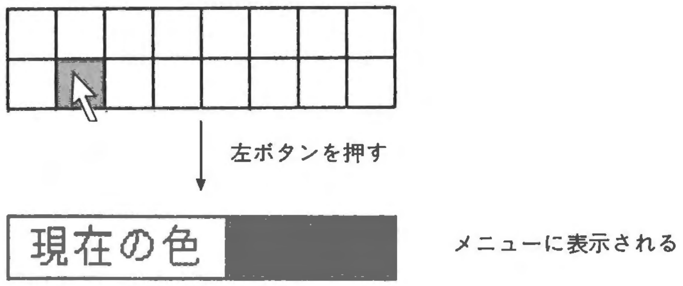
すでにペイントされている表示用ます目の上へカーソルを移動させ，左ボタンを押すと，そこのタイルパターンとデータが表示されます．データを記録し忘れた場合や，同じようなパターンを使いたい時など，非常に便利です．

6.　実行例

★色の塗り方



★色の選び方



★メニューの選び方



★カーソルの示す位置



① プログラムをRUNしたら，4 ⏎ と入力し，図のパターンを作成してタイリングを実行してください。

	青			シ			
		青			シ		
			青			シ	
				青			シ

② 次に黒を選んでから〝一色塗りつぶし〞を実行してください．全部消えましたか？　それでは，シアンで塗りつぶしてから，図のパターンを作成し，タイリングを実行しましょう．

シ	赤	赤	シ	シ	青	シ	シ
赤	赤	シ	シ	青	シ	シ	赤
赤	シ	シ	青	シ	シ	赤	赤
シ	シ	青	シ	シ	赤	赤	シ

③ セット数を増やしてみましょう．〝セット数の変更〞を選んでから，8⏎と入力してください．それから図のパターンを作成し，タイリングを実行しましょう．どうですか？先程とはだいぶ違いますね．このように，セット数が多ければ，複雑なパターンや，対称なパターンが作成しやすいのです．

				青	シ	シ	
				シ	青		
			シ			青	青
		シ				青	青
シ	シ				青		
シ	シ			青			
		シ	青				
	青	青	シ				

④ こんどは〝色変換〞をしてみましょう．③で作成したパターンの黒を灰色に変えます．まず，黒を選んでから〝色変換〞を選択してください．カーソルがカラーの中に閉じ込められましたね．それでは，左ボタンで灰色を選んでから，右ボタンで実行しましょう．どうですか？　黒が灰色に変わったでしょう．ついでにタイリングを実行して前の模様と比べてみましょう．

⑤ セット数を16にして，図のパターンでタイリングを実行してください．

				シ		青	
			シ			青	
		シ		青	青		
		シ	青		青		
		青	シ			青	
	青	赤	シ	シ		青	青
青	赤	赤	赤		シ	黄	黄
青	赤	赤	赤	シ	黄	黄	黄
青	赤	赤	シ	黄	黄	黄	黄
	青	シ		シ	黄	黄	黄
	シ	青		シ	黄	黄	青
シ			青	黄	黄	青	
			シ	青	青		シ
			シ			シ	
			シ		シ		
シ	シ			シ			

どうですか？ くじゃくの羽みたいですね．このパターンから，このようなタイリングを想像できますか？

⑥ 表示用ます目1-0のタイルセットとデータを表示してみましょう．カーソルをその場所に移動させ，左ボタンを押してください．
どうですか？ 1-0のパターンが復元できましたね．それでは，シアンを黄色に，赤をマゼンダに変更してタイリングしてみましょう．

7．データの使い方

次のようなデータを使用するとします．

```
12 10 0  12 ───→ &H12, &H10, &H0, &H12
24 20 0  24 ───→ &H20, &H20, &H0, &H24
40 40 0  48 ───→ &H40, &H40, &H0, &H48
90 80 0  90 ───→ &H90, &H80, &H0, &H90
```

```
10 TI$=CHR$(&H12,&H10,&H0,&H12,&H20,&H20,&H0,&H24,&H40
,&H40,&H0,&H48,&H90,&H80,&H0,&H90)
20 PAINT(100,100),TI$
```

この様にすればよいのです．詳しくは，4.5.2 タイリングを参照してください．

8．ボタン操作のまとめ

左ボタン
- 色の選択
- メニューの選択
- 特殊操作
- タイルセットの作成

右ボタン
- タイリングの実行
- 色変換の実行

以上で説明は終わりです．このプログラムを使って，いろいろな模様を作り出してください．また，PALETTEの色を変更して，中間色のタイリングを行うのもおもしろいでしょう．ぜひ挑戦してみてください．ただし，パレット番号の0と15は変更しないでください．

```
100 '****************************************
110 '***                                  ***
120 '***        タイリング プログラム         ***
130 '***                                  ***
140 '****************************************
150 SCREEN 0,0,0,1:COLOR 1,0:CLS:LOCATE 5,1:PRINT"このプログ
ラムの使い方は、N100-BASICユーザーズガイドに記載されて
います。":FOR I=1 TO 1000:NEXT
160 IF INP(&H22) AND 2 THEN BEEP:BEEP:BEEP:PRINT:PRINT:PRINT
"                    カラーディスプレイ及びカラーボードを使用し
てください。":BEEP:PRINT:PRINT"                    プログラムを
終了させていただきます。":END
170 '***** 初期設定 *****
180 SCREEN 1,0,0,0:WIDTH 80,25:MOUSE 0:CLS:LOCATE 1,1,0
190 NX=400:NY=51:Q=0:Z1=0:Z2=0
200 DIM D$(15),C$(7),TD$(19,2)
210 DATA 0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1
010,1011,1100,1101,1110,1111
220 FOR I=0 TO 15:READ D$(I):NEXT
230 FOR I=399 TO 719 STEP 80:LINE(I,50)-(I,500):NEXT
240 FOR I=50 TO 500 STEP 90:LINE(399,I)-(719,I):NEXT
250 A=0:FOR I=420 TO 660 STEP 80:PUT@(I,20)," "+STR$(A)+" ",
0,15:A=A+1:NEXT
260 A=0:FOR I=70 TO 430 STEP 90:PUT@(370,I),STR$(A),12,14:A=
A+1:NEXT
270 PUT@(20,402),"現在の色":PUT@(20,427),"一色塗りつぶし":PU
T@(20,452),"色変換→":PUT@(20,477),"セツト数の変更"
280 FOR I=400 TO 500 STEP 25:LINE(20,I)-(180,I):NEXT:LINE(20
,400)-(20,500):LINE(180,400)-(180,500):LINE(110,400)-(110,42
5):LINE(110,450)-(110,475)
290 FOR I=20 TO 180 STEP 20:LINE(I,350)-(I,390):NEXT:FOR I=3
50 TO 390 STEP 20:LINE(20,I)-(180,I):NEXT
300 A=0:FOR J=360 TO 380 STEP 20:FOR I=30 TO 170 STEP 20:PAI
NT(I,J),A,15:A=A+1:NEXT:NEXT
310 GOSUB 610
320 ON ERROR GOTO 1150
330 MOUSE 1,0,0,1
340 '***** MAIN ROUTINE *****
350 X=MOUSE(0):Y=MOUSE(1):CC=POINT(X-2,Y-2)
360 IF MOUSE(2,2)=1 THEN GOTO 690
370 IF X>399 AND Y>50 AND Y<500 AND MOUSE(2,1)=1 THEN GOSUB
930
380 IF X<20 OR X>179 THEN 350
390 IF Y<340 AND Y>340-20*N AND MOUSE(2,1)=1 THEN 450
400 IF Y<390 AND Y>350 THEN 510
410 IF Y<450 AND Y>425 THEN 550
420 IF Y<475 AND Y>450 AND MOUSE(2,1)=1 THEN GOSUB 1080
430 IF Y<500 AND Y>475 AND MOUSE(2,1)=1 THEN GOSUB 610
440 GOTO 350
450 IF MOUSE(2,1)=0 THEN 350
460 MOUSE 1,,,0
470 XX=INT((X-20)/20):YY=INT((Y-21)/20)
480 LINE(XX*20+21,YY*20+21)-STEP(18,18),C,BF
490 MOUSE 1,,,1
500 GOTO 350
```

```
510 IF MOUSE(2,1)=0 THEN 350
520 MOUSE 1,,,0:C=CC:LINE(111,401)-(179,424),C,BF:MOUSE 1,,,
1
530 GOTO 350
540 '***** 一色 SUB *****
550 IF MOUSE(2,1)=0 THEN 350
560 MOUSE 1,,,0:FOR I=340-(20*N)+1 TO 321 STEP 20
570 FOR J=21 TO 161 STEP 20
580 LINE(J,I)-STEP(18,18),C,BF
590 NEXT:NEXT:MOUSE 1,,,1:GOTO 350
600 '***** セツト数変更 SUB *****
610 MOUSE 1,,,0:LINE(0,0)-(350,341),0,BF
620 LOCATE 1,1,0:INPUT"セツト数 ( 1-16 ) ";N:N=INT(N)
630 IF N<1 OR N>16 THEN BEEP:GOTO 620
640 LINE(0,0)-(350,341),0,BF
650 FOR I=20 TO 180 STEP 20:LINE(I,340-20*N)-(I,340):NEXT
660 FOR I=340 TO 340-20*N STEP-20:LINE(20,I)-(180,I):NEXT
670 MOUSE 1,,,1:RETURN
680 '***** タイリング SUB *****
690 TD$(Q,2)="":TD$(Q,0)="":TILE$="":LY=17-N
700 MOUSE 1,,,0:LINE(190,0)-(310,350),0,BF:MOUSE 1,,,1
710 PUT@(210,(16-N)*20)," "+STR$(Z1)+" ",0,15:PUT@(265,(16-N
)*20)," "+STR$(Z2)+" ",12,14
720 FOR PY=330-20*(N-1) TO 330 STEP 20
730 I=0:LY=LY+1:LX=24
740 FOR PX=30 TO 170 STEP 20:C$(I)=D$(POINT(PX,PY)):I=I+1:NE
XT PX
750 FOR J=4 TO 1 STEP -1
760 K=128:TI=0
770 FOR I=7 TO 0 STEP -1:TI=TI+VAL(MID$(C$(I),J,1))*K:K=K/2:
NEXT I
780 TD$(Q,0)=TD$(Q,0)+MKI$(TI):TILE$=TILE$+CHR$(TI)
790 LOCATE LY,LX:PRINT HEX$(TI)+" ";:LX=LX+3
800 NEXT J:NEXT PY
810 FOR PY=330-20*(N-1) TO 330 STEP 20
820 FOR PX=30 TO 170 STEP 20
830 TD$(Q,2)=TD$(Q,2)+HEX$(POINT(PX,PY))
840 NEXT:NEXT
850 TD$(Q,1)=STR$(N)
860 MOUSE 1,,,0:LINE(NX,NY)-STEP(78,88),0,BF:PAINT(NX,NY),TI
LE$,15:MOUSE 1,,,1
870 NX=NX+80:IF NX>710 THEN NX=400:NY=NY+90
880 IF NY>500 THEN NY=51
890 Q=Q+1:IF Q=20 THEN Q=0
900 Z1=Z1+1:IF Z1=4 THEN Z1=0:Z2=Z2+1:IF Z2=5 THEN Z2=0
910 GOTO 350
920 '***** DATA摘出 SUB *****
930 XX=INT((X-400)/80):YY=INT((Y-50)/90):W=XX+YY*4:IF VAL(TD
$(W,1))=0 THEN BEEP:RETURN
940 IF XX=Z1 AND YY=Z2 THEN BEEP:RETURN
950 N=VAL(TD$(W,1)):MOUSE 1,,,0:GOSUB 640
960 PUT@(210,(16-N)*20)," "+STR$(XX)+" ",0,15:PUT@(265,(16-N
)*20)," "+STR$(YY)+" ",12,14
970 LY=18-N:LX=24
```

```
980 FOR J=1 TO N*8 STEP 2
990 A$=HEX$(CVI(MID$(TD$(W,0),J)))
1000 LOCATE LY,LX:PRINT A$+" ";
1010 LX=LX+3:IF LX>33 THEN LX=24:LY=LY+1
1020 NEXT J:MOUSE 1,,,0:A=1
1030 FOR I=340-(20*N)+1 TO 321 STEP 20
1040 FOR J=21 TO 161 STEP 20
1050 LINE(J,I)-STEP(18,18),VAL("&H"+MID$(TD$(W,2),A,1)),BF
1060 A=A+1:NEXT:NEXT:MOUSE 1,,,1:RETURN
1070 '***** 色変換 ＳＵＢ *****
1080 MOUSE 4,20,350,180,390:X=MOUSE(0):Y=MOUSE(1):CC=POINT(X
-2,Y-2)
1090 IF MOUSE(2,1)=1 THEN CH=CC:MOUSE 1,,,0:LINE(111,451)-(1
79,474),CH,BF:MOUSE 1,,,1
1100 IF MOUSE(2,2)=0 THEN 1080
1110 MOUSE 4,0,0,719,511:MOUSE 1,,,0:FOR PY=340-20*N+1 TO 32
1 STEP 20:FOR PX=21 TO 161 STEP 20
1120 IF POINT(PX,PY)=C THEN LINE(PX,PY)-STEP(18,18),CH,BF
1130 NEXT:NEXT:MOUSE 1,,,1:RETURN
1140 '***** ＥＲＲＯＲ 処理 *****
1150 IF ERL=860 AND ERR=5 THEN 1160 ELSE ON ERROR GOTO 0
1160 MOUSE 0:BEEP:BEEP:BEEP
1170 PUT@(190,400)," ０ ０ ０ ０が３つ以上"
1180 PUT@(190,420),"重なっているパター"
1190 PUT@(190,440),"ンは、ＰＡＩＮＴで"
1200 PUT@(190,460),"きません。"
1210 IF MOUSE(2,1)=0 THEN 1210
1220 PUT@(195,400),"                    "
1230 PUT@(200,420),"                   "
1240 PUT@(200,440),"                   "
1250 PUT@(200,460),"                   "
1260 MOUSE 1,0,0,1:RESUME 350
```

第5章
漢字処理

N_{100} - BASICの漢字処理機能としては，キーボード上の漢字キーを使用するカナ漢字変換と，BASICプログラム上で使用する種々のステートメント，関数とがあります．

漢字処理として使用する〝漢字キャラクタ〟と呼ばれるものは，JIS第一水準の漢字2965字と，特殊記号，英・数字，ひらがな，カタカナ，ギリシャ文字，ローマ字など453文字があります．これらはすべて2バイトで表現され，2バイトキャラクタと呼ばれます．これに対し，1文字を1バイトで表現する通常のキャラクタを，1バイトキャラクタまたはASCIIキャラクタと呼びます．

漢字キャラクタは，BASIC上ではREM文，PRINT文，文字定数，文字関数，ファイル名などに使用できます．この漢字キャラクタの文字列中には，1バイトキャラクタを混在させることもできます．

5.1　漢字コード

漢字キャラクタを出力するための漢字コードには，4種類あり，その指定方法は次のようになります．
例えば，〝亜〟という文字にいちて考えると各漢字コードは次のようになります．

JISコード……………3021
JIS区点コード………16区1点
内部連続コード………1666
シフトJISコード……88（1バイト目），9F（2バイト目）

また，実際に〝亜〟という漢字を出力する場合，上記の漢字コードを用いて次のようにします．

```
PRINT CHR$(&J3021)
PRINT CHR$(&K1601)
PRINT CHR$(1666)
PRINT CHR$(&H88, &H9F)
```

これら4つのどれを用いても〝亜〟という文字が出力されます．
なお，漢字キャラクタモードでは，ASCIIコードの一部を2バイトキャラクタ（漢字など）のシフトJISコードの第1バイトとして扱います．このため，漢字キャラクタモードでは&H80～&H9F，&HE0～&HFFのASCIIキャラクタ（主にグラフィックキャラクタ）が使えなくなりますので，注意してください．

以上の4つの漢字コードは，それぞれ下記のような対応となっています．詳細は，N_{100} - BASICリファレンスマニュアル　付録　漢字コード表を参照してください．

JISコード	区点コード	内部連続コード	シフトJISコード
&J2121	&K0101	256	8140
&J2122	&K0102	257	8141
⋮	⋮	⋮	⋮
&J215F	&K0132	318	817E
&J2160	&K0133	319	8180
⋮	⋮	⋮	⋮
&J217E	&K0194	349	819E
&J2221	&K0201	350	819F
⋮	⋮	⋮	⋮
&J227E	&K0294	443	81FC
&J2321	&K0301	444	8240
⋮	⋮	⋮	⋮

5.2 漢字を使用する場合の注意

BASIC上で漢字を使用する場合，8×16ピクセルのキャラクタフォントを指定したときのみ，表示が可能となります．したがってWIDTH文を用いた場合は，次に挙げる5つの組合せが有効です．

画面が横表示の場合　80(桁)×32(行)
80(桁)×32(行)
90(桁)×25(行)
90(桁)×32(行)

画面が縦表示の場合　64(桁)×45(行)

また，WIDTH USING文を使う場合はフォントサイズを8×16ピクセルに指定すれば，2バイトキャラクタの表示が可能になります．WIDTH文およびWIDTH USING文の詳細は，N_{100}-BASICリファレンスマニュアルを参照してください．

さらに，キャラクタモードの選択(グラフィックキャラクタモードまたは漢字キャラクタモード)および，カナ漢字変換をするかしないかの設定は，SCREEN文の第3パラメータと第4パラメータで行います．

例えば，

```
SCREEN ,, 0, 1
```

と指定すると，画面が一度クリアされて，漢字キャラクタモードになります．そこで漢字キーを押してください．ディスプレイ画面下部2行に，

```
[漢字] [ひら] [カナ] [英数] [記号] [コード]
=〉[                                        ]
```

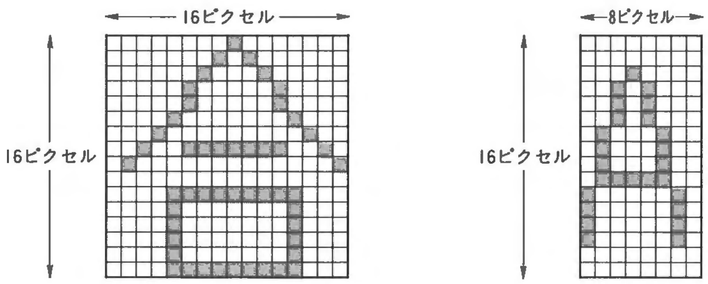
という**漢字入力用のテンプレート**が表示されます．もう一度［漢字］キーを押すと，もとの状態に戻ります．

漢字キャラクタモードの状態では，ディスプレイ画面の下部２行が，カナ漢字変換のために空けられます．さらに，ファンクションキーを表示すると，もう１行使えなくなります．LOCATE 文でカーソルの位置を指定する場合は注意が必要です．画面上の使用可能な範囲は次のとおりです．

WIDTH文 行数	LOCATE文　垂直座標	
	ファンクションキー	
	ON 時	OFF 時
25 （横） 32	22 29	23 30
45（縦）	42	43

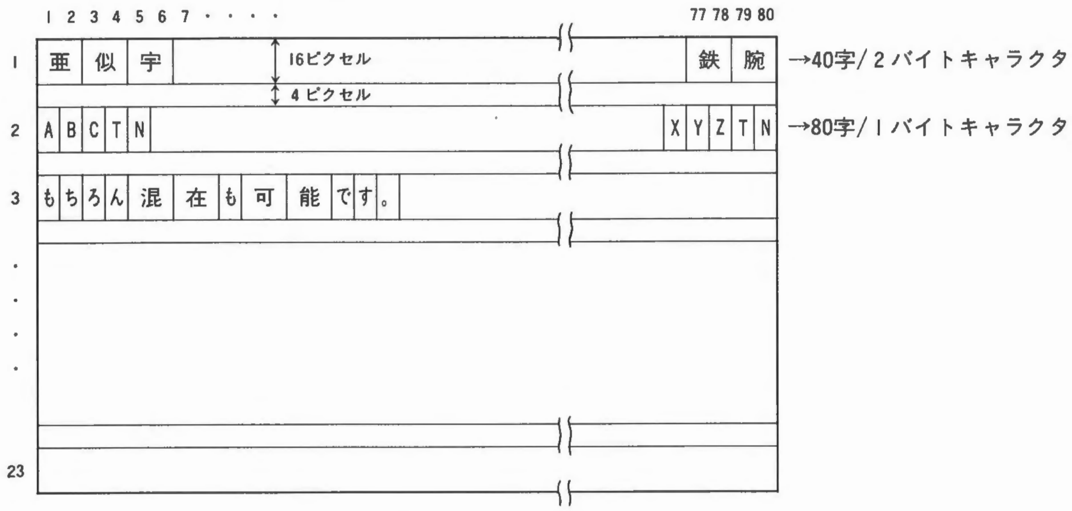
5.3　画面構成

漢字キャラクタおよびASCII キャラクタは，下のようなドットパターンで画面に表示されます．



漢字キャラクタは16×16ピクセルのドットパターンで表示されるので，BASIC で通常使われる ASCII キャラクタ２文字分のスペースを必要とします．これにより，１行中をすべて漢字キャラクタで埋めた場合は，ASCII キャラクタの半分の文字数しか表示できません．これを図に示すと次のようになります．

例) WIDTH 80, 25:SCREEN, , 0, 1　の場合



この場合，表示可能な漢字キャラクタ数は，40×23＝920文字となります．その他の場合を以下に示します．(ただし，ファンクションキーを表示しない場合)

WIDTH 文による指定		表示可能な文字数
(横表示)	80×32 の場合	1200 文字
	90×25 の場合	1035 文字
	90×32 の場合	1350 文字
(縦表示)	64×45 の場合	1376 文字(最大)

もちろん1バイトキャラクタと2バイトキャラクタの混在も可能ですので，その場合には，文字数は上の場合より多くなります．

5.4　漢字キャラクタの入出力

BASIC上で，実際に漢字キャラクタを使ってみましょう．

例えば，

```
10 REM ■
```

ここで，漢字キャラクタを使用して〝注釈です．〞という文字列を入力したい場合，まず 漢字 キーを押して漢字入力モードにします．そして，

```
[漢字] [ひら] [カナ] [英数] [記号] [コード]
=〉[ちゅう　しゃく　です。                    ]
```

と入力します．ここでの入力方法にはローマ字入力による方法と かな キーをロックして，カナ入力をする方法とがあります．

ここでリターンキーを押すと，まず最初の〝ちゅう〞が漢字変換されて，最下行に表示されます．そこでカーソルを使って〝注〞の字を選び，リターンキーを押すと画面の〝10 REM〞の後に，〝注〞の字が入力されます．

```
[漢字] [ひら] [カナ] [英数] [記号] [コード]
=〉[しゃく　です。                            ]
```

同様にしてリターンキーを押して〝釈〞の字を選びます．

```
[漢字] [ひら] [カナ] [英数] [記号] [コード]
=〉[です。                                    ]
```

次に TAB キーを押して〝かな〞モードにして，リターンキーを押すと，かなの状態のままテンプレートの文字が（漢字変換せずに）入力されます．

```
10 REM 注釈です。
```

以上で漢字キャラクタの入力を終了したので， 漢字 キーを押して漢字入力モードからぬけます．すると，現在入力している行のうしろ（〝です。〞のうしろ）にカーソルが移動しますので，リターンキーを押して入力を終了させます．このように，通常は1バイトキャラクタにより，BASICのステートメント，コマンドなどを入力し，文字列データなどに漢字が必要なときは漢字入力モードに入り，漢字キャラクタの入力を行います．漢字入力に関する詳細は，PC-100　ユーザーズガイドを参照してください．

5.5 漢字処理用関数

漢字処理用の関数として，KPOS，KLEN，JIS$，KTN$ があります．また，漢字コード入力用には，&K および&J があります．これらは，JIS$ と KTN$ の逆関数的な役割を果します．

これらの関数および&K，&J に関する詳細は，N_{100}-BASIC リファレンスマニュアルを参照してください．

BASIC 中の文字および文字列を操作する関数，ステートメントは，すべて 1 バイトキャラクタと 2 バイトキャラクタを混在して扱えるようになっています．このため，INSTR 関数が，期待どおりの動作をしない事があります．その例を下に示します．

```
PRINT INSTR("合意売得起ABCｱｶｻﾀﾅ","ｶ") ⏎ ――(1)
 15
Ok
PRINT INSTR("合意売得文ABCｱｶｻﾀﾅ","ｶ") ⏎ ――(2)
 10
Ok
```

これからわかるように，同じ文字〝カ〞を捜す場合に，うまくいく場合(1)と，うまくいかない場合(2)ができます．これは，BASIC が漢字キャラクタ（2 バイトキャラクタ）を，内部連続コードで処理するために起こるものです．つまり，(2)の文字列中の〝文〞の内部連続コードは95B6で，このうち下位バイトの B6が〝カ〞の ASCII コードと同じため，BASIC は，ここで一致した（発見した）とみなしてしまうからです．これは，調べたい文字列が 1 バイトキャラクタ 1 文字のときのみに起こり，1 バイトキャラクタが 2 文字以上，および 2 バイトキャラクタの時には生じません．この問題を解決するサブルーチンを含むプログラムリストを下に掲げます．

```
10 '
20 '   'INSTR'    PROGRAM
30 '
40 X$="合意売得文abcｱｶｻﾀﾅ"
50 Y$="ｶ"
60 N=1
70 '
80 GOSUB 120
90 PRINT X$;"  ";Y$
100 PRINT NO;" bytes"
110 END
120 '
130 IF LEN(Y$)=1 AND ASC(Y$)<256 THEN 190
140 '
150 NO=INSTR(N,X$,Y$)
160 '
170 GOTO 260
180 '
190 Y=ASC(Y$):I=N
200 C=KPOS(X$,I)
```

```
210 IF C=0 THEN 260
220 W$=MID$(X$,C,1)
230 IF ASC(W$)=Y THEN NO=I
240 IF NO=0 THEN I=I+1:GOTO 200
250 NO=KPOS(X$,NO)
260 RETURN
```

このプログラムは1バイトキャラクタが1文字のときだけ，別のサブルーチンで判断しているので，正しい結果が得られるわけです．

このプログラムでは，X$に調べる文字列を代入し，Y$に調べたい文字列を代入します．そして，何バイト目から調べるかをNに代入します(行番号40〜60)．それから，120行から始まるサブルーチンを実行します．

サブルーチン中では，Y$が1文字で1バイトキャラクタの場合と，そうでない場合の区別をします(行番号130)．そして，Y$が1文字で1バイトキャラクタの場合は，180行へ実行が移り，ここで正しい判断をします．また，そうでない場合は，INSTR関数で処理できるので150行へと実行が移ります．どちらもNOという変数に答えを代入して，250行のRETURN文でサブルーチンを終ります．その後，90行に帰りNOを表示してプログラムを終了します．

これにより，INSTR関数は正確に動作できるようになります．したがって，INSTRで1バイトキャラクタ1文字の判断をするときは，このプログラムの180行から240行を参考にして，プログラムを作成してください．

さて，INSTR関数ではその文字（調べる文字列）が，左から何バイト目にあるかを返すものですが，〝何文字目にあるか〞を得たい場合があります．特に2バイトキャラクタを使用していると，INSTR関数では，バイト数＝文字数　が成立しないので文字列操作がやっかいなものになります．そこで，〝拡張INSTR関数〞とでも言うプログラム（サブルーチン）を掲げておきます．

```
10 '
20 '   '拡張INSTR'    PROGRAM
30 '
40 X$="合意売得文abcｱｶｻﾀﾅ"
50 Y$="ｶ"
60 N=1
70 '
80 GOSUB 120
90 PRINT X$;"   ";Y$
100 PRINT NO;" characters"
110 END
120 '
130 IF LEN(Y$)=1 AND ASC(Y$)<256 THEN 200
140 '
150 NO=INSTR(N,X$,Y$):IF NO=0 THEN 260
160 C$=LEFT$(X$,NO-1)
170 C=LEN(C$)-KLEN(C$) : NO=NO-C
180 GOTO 260
190
200 Y=ASC(Y$):I=N
210 C=KPOS(X$,I)
220 IF C=0 THEN 260
230 W$=MID$(X$,C,1)
240 IF ASC(W$)=Y THEN NO=I
250 IF NO=0 THEN I=I+1:GOTO 210
260 RETURN
```

これにより，NOという変数に文字数(1バイトキャラクタと2バイトキャラクタ混在のキャラクタ数)を返すことができます.プログラムの内容については，ほとんどINSTR関数のプログラムと同じなのでそちらを参照してください.

第6章

データファイル入出力

データファイルには，シーケンシャルファイルとランダムファイルの２種類があります．シーケンシャルファイルとはデータをファイルの先頭から順に読み書きするファイルです．一方，ランダムファイルはデータを任意の順に読み書きできるファイルです．

シーケンシャルファイルとランダムファイルには一長一短があり，どちらが良いとは一概に言えません．ただ，高速処理を要求される場合やデータの更新が頻繁に行われる場合はランダムファイルにした方がよいでしょう．一方，ファイル領域を小さくしたい場合やデータの更新があまりない場合はシーケンシャルファイルがよいでしょう．

シーケンシャルファイルとランダムファイルでは，使用できる文と関数は次のとおりです．

	シーケンシャルファイル	ランダムファイル
文	OPEN CLOSE	
	PRINT # PRINT # USING INPUT # LINE INPUT # WRITE #	FIELD LSET RSET PUT GET
関数	INPUT ¥ INPUT $ EOF	LOC LOF

6.1　シーケンシャルファイル

シーケンシャルファイルのデータを読み書きするときは一般に次の手順に従います．

① ファイルをオープンする．
② ファイルのデータを読み書きする．
③ ファイルをクローズする．

つまり，データを読み書きする前にファイルをオープンし，読み書きが終ったならば，ファイルをクローズします．またシーケンシャルファイルはオープン時に，次のいずれかの処理形態（モード）を指定します．

- OUT PUT モード：新規にファイルを設定して，データを順次書き込む．
- APPEND モード：既存のファイルの最後にデータを追加する．
- INPUT モード：既存のファイルの先頭からデータを順次読み込む．

6.1.1　シーケンシャルファイルの作成

シーケンシャルファイルを新規に作成する場合は次の手順に従ってください.

①、ファイルをOUTPUTモードでオープンする.

OPEN "TEL" FOR OUTPUT AS #1

② ファイルにデータを書き込む.

WRITE #1, N$; TN$

PRINT #1, N$; ", "; TN$

③ 最後にファイルをクローズする.

CLOSE #1

次のプログラムはキーボードから名前と電話番号を読み込んで，電話帳ファイル "TEL" を作成するものです.

```
10 REM MAKE TEL
20 OPEN "TEL" FOR OUTPUT AS #1
30 INPUT "NAME";N$
40 IF N$="" THEN 80
50 INPUT "TEL NO.";TN$
60 WRITE #1,N$;TN$
70 GOTO 30
80 CLOSE #1
```

60行は次のようにPRINT#文で書いてもかまいません.

```
60 PRINT #1,N$;TN$
```

このプログラムを実行してみましょう.

```
run
NAME? ｼﾞﾛｳ
TEL NO.? 245-4567
NAME? ﾀﾛｳ
TEL NO.? 222-3453
NAME? ﾊﾅｺ
TEL NO.? 345-5612
NAME?
Ok
```

この結果次のようなファイル "TEL" が作成されます.

ジロウ，245―4567 CR LF	タロウ，222―3453 CR LF	ハナコ，345―5612 CR LF

ここで，キャリッジリターンコード＋ラインフィードコードはPRINT #文あるいはWRITE #文の終りで自動的に書き込まれます．そして，キャリッジリターンコード＋ラインフィードコードまでのデータをレコードと呼んでいます．一般に，シーケンシャルファイルではレコードの長さはまちまちです．このようなレコード形式を可変長レコード形式と呼んでいます．

また，ファイルの読み書きはすべてバッファという入出力のためのメモリ領域を介して行われます．つまり，OPEN文，WRITE文，CLOSE文などで指定するファイル番号#nは，このバッファの番号を指しているわけです．このファイル番号の数は，BASIC起動時に／F：パラメータによって指定された範囲内であれば，いくつでも指定することができます．なお，詳細は8章　BASICの特殊な起動方法を参照してください．

PRINT #文とWRITE #文のどちらを使っても，シーケンシャルファイルにデータを書き込むことができますが，WRITE #文は区切り記号カンマ(,)を自動的にデータ間に書き込み，余分な空白を削除します．

また，PRINT # USING文を使って，編集されたデータを書き込むこともできます．たとえば，

```
PRINT #1, USING "####.##",A;B;C;D
```

とすると，A，B，C，Dの値は書式指定に従って編集されてファイルに書き込まれます．

なお，OUTPUTモードでオープンすると，すでに同じ名前のファイルがあった場合，それを削除して新たにつくりなおすことになります．

6.1.2 シーケンシャルファイルへのデータの追加

既存のシーケンシャルファイルへデータを追加する場合は，次の手順で行います．

① ファイルをAPPENDモードでオープンする．

```
OPEN "TEL" FOR APPEND AS #1
```

② ファイルに追加したいデータを書き込む．

```
WRITE #1, N$; TN$
PRINT #1, N$; ","; TN$
```

③ 最後にファイルをクローズする．

```
CLOSE #1
```

オープン時のモードをAPPENDとする以外はシーケンシャルファイルを作成する手順とまったく同じです．APPENDモードでオープンするファイルがディスクに作られていないとエラーとなります．

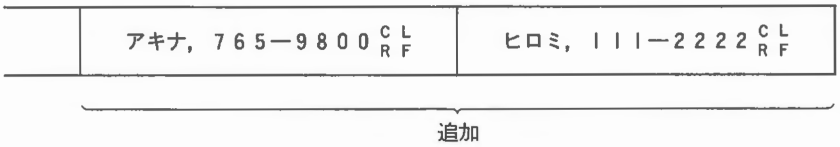
では，先程作成したファイル"TEL"に新たなレコードを追加するプログラムを作成してみましょう．

```
10 REM MAKE TEL
20 OPEN "TEL" FOR APPEND AS #1
30 INPUT "NAME";N$
40 IF N$="" THEN 80
50 INPUT "TEL NO.";TN$
60 WRITE #1,N$;TN$
70 GOTO 30
80 CLOSE #1
```

このプログラムは先のプログラムの20行を APPEND モードに変えただけです．ではプログラムを実行してみましょう．

```
run
NAME? ｱｷﾅ
TEL NO.? 765-9800
NAME? ﾋﾛﾐ
TEL NO.? 111-2222
NAME?
Ok
```

この結果，シーケンシャルファイル “TEL” には次のようにデータが追加されます．



6.1.3 シーケンシャルファイルのデータの読込み

シーケンシャルファイルからデータを読み込むときは次の手順で行います．

① ファイルを INPUT モードでオープンする．

```
OPEN "TEL" FOR INPUT AS #1
```

② ファイルからデータを読み込む．

```
INPUT #1, N$, TN$
```

③ ファイルをクローズする．

```
CLOSE #1
```

次のプログラムはファイル “TEL” からすべてのデータを読み込み，それを画面に表示するものです．

```
10 REM READ TEL
20 OPEN "TEL" FOR INPUT AS #1
30 IF EOF(1) THEN 70
40 LINE INPUT #1,R$
50 PRINT R$
60 GOTO 30
70 CLOSE #1
```

40行の INPUT # 文に，ファイル “TEL” の先頭から 1 レコードずつ順に読み込んでいきます．このとき，N$には名前が，TN$には電話番号がセットされます．

30行のEOF関数はファイルの終りを検出する関数です．これを省略すると，ファイルの最後に達したとき，“Input　past　end”エラーが発生します．

このプログラムを実行すると次のように表示されます．

```
run
ｼﾞﾛｳ              245-4567
ﾀﾛｳ               222-3453
ﾊﾅｺ               345-5612
ｱｷﾅ               765-9800
ﾋﾛﾐ               111-2222
Ok
```

INPUT＃文でデータを読むときには次の記号が区切り記号となります．

空白　　カンマ(,)　キャリッジリターンコード(CR)

ただし，文字データを読み込む場合，空白は区切り記号とはなりません．

キャリッジリターンコード＋ラインフィードコードのみを区切り記号としたい場合は，LINE INPUT #文を使います．次のプログラムはLINE　INPUT #文を使ってファイル“TEL”を読み込むものです．

```
10 REM READ TEL
20 OPEN "TEL" FOR INPUT AS #1
30 IF EOF(1) THEN 70
40 INPUT #1,N$,TN$
50 PRINT N$,TN$
60 GOTO 30
70 CLOSE #1
```

```
run
"ｼﾞﾛｳ","245-4567"
"ﾀﾛｳ","222-3453"
"ﾊﾅｺ","345-5612"
"ｱｷﾅ","765-9800"
"ﾋﾛﾐ","111-2222"
Ok
```

特に区切り記号を設けず，バイト数によってデータを読み込む関数として，INPUT$があります．ファイル〝TEL″の最初の6バイトを読み込むプログラムは次のようになります．

```
10 REM READ TEL
20 OPEN "TEL" FOR INPUT AS #1
30 R$=INPUT$(6,1)
40 PRINT R$
50 CLOSE #1

run
"ｼﾞﾛｳ"
Ok
```

6.2 ランダムファイル

ランダムファイルのデータを読み書きするときはバッファの存在をかなり意識しなければなりません．具体的には次のようになります．

① ファイルをオープンする．
② バッファのフィールドを定義する．
③ バッファを経由してデータを読み書きする．
④ ファイルをクローズする．

6.2.1 ランダムファイルの作成

ランダムファイルを作成するには次の手順に従います．

① ファイルをオープンする．

```
OPEN "XTEL" AS #1
```

ランダムファイルをオープンするときはモードの指定は不要です．

② フィールドを定義する．

```
FIELD #1, 10 AS NF$, 20 AS TNF$
```

③ バッファにデータをセットする．

```
LSET  NF$= "ヤマダ"
RSET  TNF$= "680-1921"
```

バッファにデータをセットするのは代入文ではできません．LSET または RSET を使わなければなりません．LSET は左づめ，RSET は右づめにして，それぞれバッファにデータをセットします．

④ バッファのデータをディスクに書き込む.
PUT　#1, R

ランダムファイルではレコードの長さはふつう128バイトで,各レコードには1から順に番号が振られます.この番号のことをレコード番号と呼び,ランダムファイルのレコードを読み書きするときには必ずこのレコード番号を指定しなければなりません.
なお,レコードの長さは,BASICの起動時に/S:パラメータによってファイルバッファサイズを指定することにより,任意に変えることができます.詳細は8章　BASICの特殊な起動方法をご覧ください.

⑤ ファイルをクローズする.

CLOSE　#1

次のプログラムは電話帳をランダムファイルとして作成するものです.

```
10 REM MAKE XTEL
20 R=1
30 OPEN "XTEL" AS #1
40 FIELD #1,10 AS NF$,20 AS TNF$
50 INPUT "NAME";N$
60 IF N$="" THEN 130
70 INPUT "TEL NO.";TN$
80 LSET NF$=N$
90 RSET TNF$=TN$
100 PUT #1,R
110 R=R+1
120 GOTO 50
130 CLOSE #1
```

実行してみましょう.

```
run
NAME? ﾖｼｺ
TEL NO.? 333-4444
NAME? ﾄｳｷﾁﾛｳ
TEL NO.? 567-5678
NAME? ﾅﾅ
TEL NO.? 492-0176
NAME?
Ok
```

この結果次のようなランダムファイル“XTEL”が作られます.

	10バイト	20バイト	226バイト
レコード 1	ヨシコ	333—4444	空き
〃 2	トウキチロウ	567—5678	空き
〃 3	ナナ	492—0176	空き

なお，OPEN 文で指定したファイルがディスク上にないと，新たにファイルがつくられますが，ファイルがディスク上にあれば，既存のファイルに，追加または重ね書きされることになります．

また，ランダムファイルの読み書きもすべてバッファを介して行われ，OPEN 文，PUT 文，GET 文などで指定されるファイル番号はこのバッファ番号を意味します．ファイル番号の数についてはシーケンシャルファイルの場合とまったく同様です．

6.2.2 ランダムファイルからのデータの読込み

ランダムファイルからデータを読み込むときは次の手順で行います．

① ファイルをオープンする．

```
OPEN "XTEL" AS #1
```

やはり，モード指定は不要です．

② バッファのフィールドを定義する．

```
FIELD #1, 10 AS NF$, 20 AS TNF$
```

通常，作成したときと同じように定義します．

③ ディスクのデータをバッファに読み込む．

```
GET #1, R
```

レコードを読み込む場合，読み込むレコードの番号を指定します．

④ バッファのデータを取り出す．

```
A$=NF$
```

あるいは，

```
PRINT NF$
```

バッファのデータを取り出すには，代入文を使います．単に表示させたい場合なら，PRINT 文で直接指定してもかまいません．

⑤ ファイルをクローズする．

```
CLOSE #1
```

次のプログラムは“XTEL”のデータを読み込み，表示します．

```
10 REM READ XTEL
20 OPEN "XTEL" AS #1
30 FIELD #1,10 AS NF$,20 AS TNF$
40 NMAX=LOF(1)
50 FOR R=1 TO NMAX
60 GET #1,R
70 PRINT NF$,TNF$
80 NEXT R
90 CLOSE #1
```

ここでLOF関数は，ファイルの最大レコード番号を得る関数です．
これを実行すると次のように表示されます．

```
run
ヨシコ                      333-4444
トウキチロウ                567-5678
ナナ                        492-0176
OK
```

6.3　数値データの読み書き

バッファ内のデータはすべて文字列として扱います．したがって，数字データをバッファにセットする場合，あるいはバッファの内容を数値変数に代入する場合，それぞれ文字型，数値型への変換が必要となります．そのために用意されている関数を次にあげます．

数値型から文字型へ

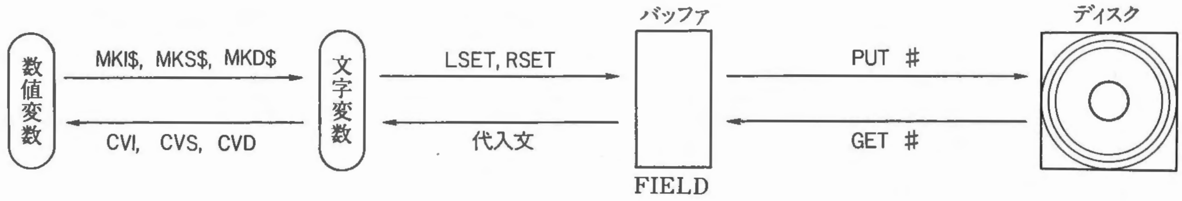
MKI$(N%)　　整数を2バイトの文字列へ変換
MKS$(N)　　単精度数値を4バイトの文字列へ変換
MKD$(N#)　　倍精度数値を8バイトの文字列へ変換

文字型から数値型へ

CVI(N$)　　2バイトの文字列を整数へ変換
CVS(N$)　　4バイトの文字列を単精度数値へ変換
CVD(N$)　　8バイトの文字列を倍精度数値へ変換

例）
```
I=100
X$=MKI$(I)
N=CVI(X$)
```

ただし，X$はフィールド名です．



6.4 ファイルスペックについて

OPEN 文で開くファイルの名前を指定するとき，ドライブ名やパス名を含めた〝ファイルスペック〟として指定することにより，任意のドライブ中の，任意のディレクトリ中のファイルに対して入出力を行うことができます．

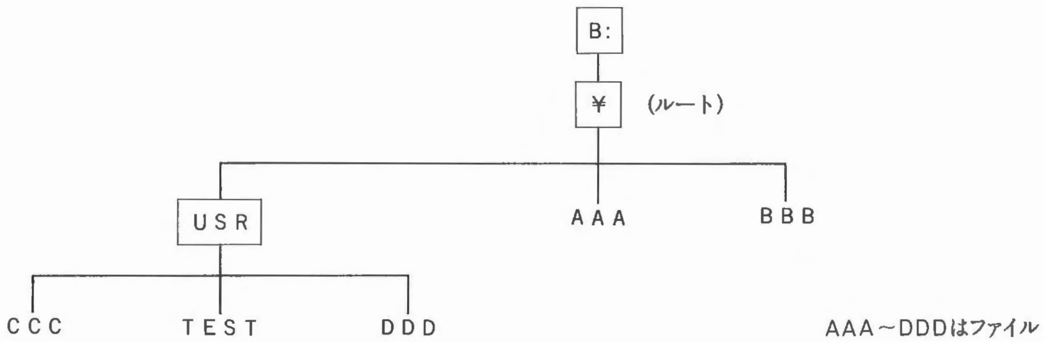
ファイルスペックは，次のような構成となっています．

〔<ドライブ名>：〕〔<パス名>〕<ファイル名>

ドライブ名が省略されたときには，現在 DOS 上で選択されているドライブ（カレントドライブ）が仮定されます．

パス名は<ファイル名>で指定されるファイルが格納されているディレクトリ，あるいはこれから<ファイル名>で指定されるファイルを格納しようとするディレクトリを，正しく指し示すためのものです．

たとえば，ドライブ B：のルートディレクトリ（ドライブごとにある最も上位にあるディレクトリ）の下に〝USR〟というディレクトリがあり，そこに〝TEST〟というファイルがあるとします．



この〝TEST〟ファイルを入力モードで開くには，次のようにします．

```
OPEN "B:¥USR¥TEST" FOR INPUT AS #1
```

ファイルスペック〝B：¥USR¥TEST〟中，〝¥USR¥〟がパス名です．

もしディレクトリ〝USR〟中に新しくファイルを作る場合にも，同じようにパス名を指定すればよいのです．

```
OPEN "B:¥USR¥NEW" FOR OUTPUT AS #2
```

これで〝NEW〟ファイルがディレクトリ〝USR〟中に新しく開かれます．

ディレクトリが階層的に多数重なっているときは，パス名を¥で区切って順に並べます．

```
OPEN "A:¥USR¥TARO¥ZZZ" AS #1
```

この例は，ドライブＡ：のルートディレクトリの下にある"USR"ディレクトリの，さらに下にある"TARO"ディレクトリ中に "ZZZ" というファイルをランダムモードでオープンするものです．

なお，下位のディレクトリになればなるほど，アクセスに要する時間は長くなりますので，速度を要求される場合は，ファイルをなるべくルートディレクトリに近いディレクトリ中にオープンすることをお勧めします．

ファイルの階層構造，パス名などについての詳細は10章　ファイルの階層構造をご覧ください．

第7章
デバッグの基本

7.1　デバッグとは

バグを出してしまう第一の原因は，そのプログラムで何をしようとしているのか明確にせずにプログラミングを始めてしまうことにあります．人間に命令するなら「そこをああしてこうして……」ですむことも，コンピュータを相手にするときは，「そこ」とは何か，「ああして」とは何か，一つひとつをBASICという言語で表現できなければなりません．プログラムを作るときはまず，フローチャート（流れ図）を書いて，どういう処理をしてどういう結果を得たいのかを明確にしておいてください．

次に，コンピュータにさせたいことははっきりしても，慣れないうちはそれをBASICで表現するのは困難なことがあります．また，実際にキーボードからプログラムを入力するときに間違っていることもあるでしょう．こうした初歩的なミスを発見するためにBASICにはさまざまなエラートラップ機能があります．プログラムの実行中に，実行できない命令があると〝Syntax error in 10〟のようにメッセージが出力されて実行が中断されます．エラーメッセージの意味は〝BASICリファレンスマニュアル付録　エラーメーセージ〟にまとめてありますので，これとプログラムのリストを見くらべてみれば，つまらない間違いはまず取り除くことができます．

〝Syntax error〟と〝Illegal function call〟は特に頻出するエラーです．特に〝Illegal function call〟はエラーの出た行以外にその原因があることが多く，修正が困難です．例えば次のようなものです．

```
10 DIM A(10)
20 FOR I=0 TO 10
30 A(I)=I
40 NEXT
50 PRINT"X ハ゛ンメイコウノ テ゛ータヲ マエニ 3ツス゛ラシマス"
60 INPUT"X=";X
70 FOR I=X TO 10
80 A(I-3)=A(I)
90 NEXT
```

Xの値として2以下を入れるとエラーが80行で出ますが，この行は間違っていません．これは変数Iが3以下にならないように，60行あたりを工夫する必要があります．

〝Syntax error〟の方も次のような2つのケースに限ってエラー発生の行自体は正しいということがあります．

```
10 READ A
20 PRINT A
30 DATA B
```

エラーは30行で出ますが，その原因は10行の READ 文で数値を読もうとしていることによる場合があります．データは文字型の〝B″になっているからです．

もう一つは

```
10 DEF FNA(B,C)=5*(B+(C-2)
20 PRINT FNA(5)
```

のような場合です．

エラーは20行で出ますが，実際の Syntax error は10行（かっこが1つ足りない）にあります．

7.2 PRINT,STOP,INPUT$を使ってのデバッグ

初歩的なデバッグでは，これらの命令をデバッグのためにプログラム中に入れておくことがよくあります．

例えば10個の乱数（0～9の整数）を配列に代入し，ゆっくりと表示しようとしたとします．

```
10 DIM A(10)
20 FOR I=1 TO 10
30 A(I)=INT(RND*10)
40 PRINT A(I)
50 GOSUB *WAITING
60 NEXT
70 END
80 *WAITING
90 FOR I=0 TO 1000:NEXT
100 RETURN
```

RUN してみると，数値を1つ表示して終わってしまいます．気が付いてみれば本当に簡単なミスなのですが，これがもっと複雑なプログラムの一部だと，何時間も「何故だろう」と頭をひねることが少なくありません．

このようなプログラムのバグは例えば次のように見つけます．

(1)プログラムの実行によって変数がどう変化しているか，PRINT 文で確かめてみる

先程のプログラムの実行終了後，PRINT　I をダイレクトモードで実行すると1002になっています．配列Aの内容を1番目から10番目まで表示するつもり（20～60行）が1002番目というとんでもない値になっています．これは90行で，たかが時間かせぎのループと思ってあまり注目しなかった FOR～NEXT ループで，ループカウンターに20行と同じ I を使ってしまったことが原因とわかります．90行のループが終わると I は1001になり，サブルーチンから戻ると60行の NEXT でさらに＋1されます．これで20行で宣言した FOR 文のカウンタ I は10を超えてしまい，ループが1回で終わってしまうのです．

ループカウンタが不本意に変化していないことを確認するために，

```
55 PRINT I
```

のような行を一時的に挿入しておくことも，よく行われます．この他，〝Illegal function call″，〝Subscript out of range″などのエラーが出たときも，まずはエラーの出た行の変数の値をプリントアウトしてみることがデバッグの第一歩です．

(2) STOP 文を挿入してみる

10個の整数型の乱数を発生し，小さい順に並べかえて表示するプログラムを考えてみます．

1番目のデータを最小と仮定し，2番目，3番目…と比較してゆき，より小さいものがあったら1番目と交換します．10番目まで比較したら，最小のものは1番目に入っているので，次は2番目を2番目に小さいものとして3〜10番目と比較して交換するという作業を繰り返すというアルゴリズムを使います．

```
10 DEFINT A-Z
20 DIM A(10)
30 FOR I=1 TO 10
40 A(I)=RND*10
50 PRINT USING"####";A(I);
60 NEXT
70 PRINT:PRINT"SORT!"
80 FOR I=1 TO 10
90 MIN=A(I)
100 FOR J=1 TO 10
110 IF A(J)<MIN THEN SWAP A(J),A(I):MIN=A(J)
120 NEXT
130 NEXT
140 FOR I=1 TO 10
150 PRINT USING"####";A(I);
160 NEXT
170 END
```

さてこのプログラムを実行してみると，何か少し変です．そこで次の行を追加してみてください．

```
115 PRINT I;J;MIN:STOP
```

こうすると一回比較するごとに配列の何番目と何番目を比較して最小の値がどうなっているか知ることができます．配列の元データは表示されていますので，それをこのアルゴリズムで並べ換えるとどうなってゆくかわかるはずです．途中で矛盾がなければ，CONT コマンドで続きをやってみます．こうしているうちに MIN の値が正しくないことに気づけば，110行の MIN＝A（J）は MIN＝A（I）の間違いであることがわかるはずです．

このような場合，更に便利な関数 INPUT$を使うこともできます．

(3) INPUT$で実行速度を操る

コンピュータの演算は非常に速いので，どのようにプログラムが進行しているのかわかりにくいことがあります．STOP 文を使えば1ステップごとに止めることもできますが，いちいち Break in…とメッセージがでたり，CONT を入力しなければならないのもわずらわしいものです．そこで先程の115行を次のように替えてみてください．

```
115 PRINT I;J;MIN:A$=INPUT$(1)
```

こうすると何かキーを押すごとに作業が一歩一歩進んでゆきます．ループのあるプログラムで，何回目かにエラーがでるようなもの，画面処理のエラートラップなどには最適な方法でしょう．変数がループの中で，どう変化してゆくか，思いのままの実行速度で手に取るようにわかります．画面をあまり壊したくないときなどはPRINT文の前にLOCATE文を入れておくとよいでしょう．

7.3 TRON／TROFFを使ってのデバッグ

プログラム実行の流れがどうなっているかわからず，エラーになってしまうことがあります．このようなときに力を発揮するのがTRONコマンドです．次のプログラムをみてください．

```
10 A=1/3
20 B=1/3
30 C=1/3
40 IF A+B+C=1 THEN 70
50 PRINT"1/3+1/3+1/3<>1? ｿﾝﾅﾊﾞｶﾅ!"
60 END
70 PRINT"1/3+1/3+1/3=1 ﾆ ｷﾏｯﾃｲﾙ"
80 END
```

1/3を3回足せば1になりそうなものですが，コンピュータの内部では1/3のような小数は正確に記憶できないので僅かな誤差が伴い，A＋B＋C＝0.9999…のようになり40行のIF文は成立しません．従ってプログラムの実行は70行へは移りません．TRONコマンドをダイレクトモードで実行してからRUNすると…

```
TRON
Ok
RUN
[10][20][30][40][50]1/3+1/3+1/3<>1? ｿﾝﾅﾊﾞｶﾅ!
[60]
Ok
```

確かに40行の次は50行に制御が移っています．

このようにプログラムのある部分が実行されているか，予想外の行に制御が移っていないかを知るためにはTRONコマンドを使うのです．このコマンドはプログラム実行終了後も有効です．必要でなくなったらTROFFを忘れてはなりません．

7.4 エラー処理ルーチン

エラー処理ルーチンはON ERROR GOTO文で宣言するもので，一般的な使い方は，プログラム実行中におけるユーザーの入力ミスを防ぐ，エラーの特殊なケースを処理する，などで例えば次のようなものです．

```
10 ON ERROR GOTO 100
20 INPUT X
30 PRINT"SQR(X)=";
40 PRINT SQR(X)
50 GOTO 20
100 X=-X
110 PRINT"i";
120 RESUME
```

これで負の値の根も虚数ｉｎとして表示できるのですが，デバッグ中にはとんでもないエラーの原因となることがあります．例えば20行を

```
20 INPUT "X"
```

としてからRUNしてみると20行と100行以下で無限ループができてしまいます（TRONコマンドを利用するとよくわかる)．これは20行でSyntax errorが発生したのに100行以下のエラー処理ルーチンではXが負だから，正の値にしてやれば良いという見当違いの手当をして同じ箇所に戻るので，ループになってしまうのです．ON ERROR GOTO文を使うプログラムは初め，

```
10 'ON ERROR GOTO 100
```

のようにして十分試行してみて，予想外のエラーがでないことを確認してから使った方が安全です．また処理ルーチンの先頭で

```
100 IF ERR=_ AND ERL=_ THEN 処理 :RESUME
110 PRINT ERR:PRINT ERL
120 END
```

のようにしておけば予定のエラーは処理し，それ以外は修正するために，実行を中止することができます．特殊なケースでエラーのでるようなプログラムでは有効な方法でしょう．このとき，ERRコードは数字ですが，ERROR ERRを実行すればそれに対応するエラーメッセージを出すことができます．また120行のEND（ON ERROR GOTO 0でもよい）文を省略すると〝No RESUME〟エラーになり，デバッグの役目を果せなくなってしまいます．

/I

/S：および/F：スイッチを有効にします．/Iが指定されていないと，/S：，/F：両スイッチはその効力を失います．

N_{100}-BASICは，ファイル操作を行なうために必要な領域を，ダイナミックに割り付けます．このため，N_{100}-BASICでは本来/S：および/F：スイッチをサポートする必要がありません．しかし，BASICプログラムによっては，データ構造が静的（スタティック）であるように書かれているものもあります．こうしたBASICプログラムとのコンパチビリティを保つため，N_{100}-BASICでは，/Iスイッチを指定したときに限り，/S：および/F：スイッチが有効となり，ファイル操作のための領域をスタティックに割り付けることができるようになっています．

/F：〈ファイル数〉

このスイッチは，/Iスイッチがコマンド行に指定されていなければ，無視されます．

このスイッチと/Iスイッチが指定されると，BASICプログラムの実行中同時に開くことのできる最大ファイル数が定められます．各々のファイルは，ファイルコントロールブロック（FCB）用に62バイト，ファイルバッファ用に128バイト必要とします．このファイルバッファのサイズは/S：スイッチによって変えることができます．/F：スイッチが指定されていない場合，ファイル数には3がセットされます．

MS-DOSのサポートするオープン可能なファイル数は，CONFIG.SYSファイル中の〝FILES＝〟パラメータによって指定された数により決まります．BASIC用としては〝FILES＝10〟を指定しておくとよいでしょう．最初の3ファイルは，標準入出力デバイス（キーボードおよびディスプレイ画面）用やプリンタ用として使われます．さらに1ファイルがLOAD，SAVE，CHAIN，NAME，MERGE各文の処理用として使われます．したがって，残りの6ファイルをユーザー用のファイルとして使用することができます．すなわち，CONFIG.SYSファイル中に〝FILE＝10〟が指定されているときには，/F：6が，MS-DOSでサポートできる最大ファイル数となるわけです．

すべてのファイルを使い果して（オープンして）から，さらにファイルを開こうとすると，〝Too many files〟エラーが生じます．

/S：〈ファイルバッファサイズ〉

このスイッチは，/Iスイッチがコマンド行に指定されていなければ，無視されます．

このスイッチと/Iスイッチが指定されると，ランダムファイルで使用されるファイルバッファサイズが定められます．

OPEN文中のレコードサイズ指定用オプションは，この値を超してはなりません．/S：オプションが省略された場合，デフォルトのレコードサイズは128バイトとなります．

/M：〔〈メモリの上限〉〕〔，〈最大ブロックサイズ〉〕

このスイッチを指定すると，BASICによって使用されるメモリの上限アドレスがセットされます．機械語サブルーチンをBASICプログラムと共に使用したい場合，この/M：スイッチによりBASICが使用するメモリの上限をセットする必要がある場合があります．このスイッチが省略されるか，〈メモリの上限〉に0が指定されると，BASICの使用できる最大値&HFFFF（65535）が上限としてセットされます．

メモリの上限より上の領域に何かロードしたい場合は，それらの領域を確保するためのパラメータ〈ブロックサイズ〉を使います．

〈ブロックサイズ〉は，パラグラフ（16バイトの倍数）単位でなくてはなりません．省略時には，&H1000（4096）が想定され，これによって65536（=4096＊16）バイトがBASIC用のエリアとして割り付けられます．

例えば，65536バイトをBASICに，512バイトを機械語サブルーチンに割り当てたい場合には，〝/M：，&H1010〃（BASICに4096パラグラフ，機械語サブルーチンに16パラグラフの意）を指定します．

またこのオプションは，SHELL文によってロードされる他のプログラム用にメモリを開放するために，BASIC用のブロックを小さくするように使うこともできます．

〝/M：，2048〃は，データおよびスタック用に最大32768バイトの領域を割り付ける，ということになります．

〝/M：32000，2048〃は，最大32768バイトを割り付けるが，BASICが使用するのは下の32000バイトであり，残りの768バイトは，ユーザーが自由に使用できる，という意味となります．

/C：〈通信バッファサイズ〉

このスイッチは，RS232C通信回線用の受信バッファサイズを設定するものです．〈通信バッファサイズ〉に0を指定すると，RS232C通信回線はサポートされなくなります．

/C：スイッチが省略されると，受信バッファのサイズは256バイトとなります．

例）　BASIC PAYROLL

3個のファイルを使用する環境下で，PAYROLL.BASをロードし，実行します．

BASIC INVENT/F：6

6個のファイルを使用する環境下で，INVENT.BASをロードし，実行します．

BASIC/F：4/S：512

4個のファイルの使用を許し，ランダムファイルのレコードサイズを512バイトとします．

BASIC TTY/C：512

3個のファイルを使用する環境下で，RS232受信バッファを512バイトに，送信バッファを128バッファに割りつけ，TTY.BASをロードし，実行します．

BASIC/C：0/M：32768

RS232C通信回線のサポートは行わず，BASICははじめの32kバイトだけ使用します．したがって32kバイトを超える領域はユーザーに開放されますので，機械語サブルーチンなどをここに置くことができます．

8.2　入出力のリダイレクト

BASICのコマンド行に，入力ファイル名，出力ファイル名を指定することにより，標準の入力，出力デバイスであるキーボード，ディスプレイ画面を，それぞれ他のデバイスあるいはファイルにおきかえることができます．

BASICのコマンド行を示します．ここではスイッチ類を省略してあります．

BASIC 〔〈プログラムファイル名〉〕
〔<〈入力ファイル名〉〕〔〔>〕> 〈出力ファイル名〉〕

〈プログラムファイル名〉には，BASIC起動後に実行させるプログラムの名前を指定します．

指定されたプログラム内に，キーボードからの入力（INPUT文，LINE INPUT文，INKEY$関数，INPUT$関数，INPUT¥関数）あるいはディスプレイ画面への出力（PRINT文，PRINT USING文，WRITE文）に関するステートメントや関数が含まれている場合，後述のリダイレクトによる入出力が行われることになります．

〈入力ファイル名〉を指定する場合は，その前に〝<〟をつけ加えます．このリダイレクト入力ファイルが指定されると，BASICの起動と共に，前述の〈プログラムファイル名〉によって指定されたプログラムがロードされ，実行が開始されたときに次のような現象が生じます．

- プログラム内に，

  INPUT
  LINE INPUT
  INKEY$
  INPUT$
  INPUT¥

  の各文があると，キーボード入力の代りに，〈入力ファイル名〉で指定されたリダイレクト入力ファイルからデータを読み込みます．このとき，BASICは〈CTRL-Z〉コード（CHR$（&H1A））を検出するまで，そのファイルからデータを読み続けます．ファイルが〈CTRL-Z〉コードで終結されていなかったり，プログラム内の上記の各文がファイルの終端を過ぎた後をさらに読み進もうとしたりした場合，開かれているすべてのファイルは閉じられ，〝Read past end〟エラーがディスプレイ画面に出力されて，BASICはDOSに戻ります．

  逆に，プログラム内の上記の各文により読み込むデータの数よりリダイレクト入力ファイル中のデータの方が多かった場合は，余ったデータはBASICに対するダイレクトモードでの入力とみなされます．

- プログラム内で，キー割込み処理が定義されている場合は，リダイレクトの入力ファイルに関係なく，キーボードからの入力によるキー割込みはその効果を失いません．

- 〈入力ファイル名〉に〝KYBD：〟を指定した場合は，キーボードから入力されるデータが読み込まれます．

- リダイレクト入力ファイルの指定がされたときは，プログラムの実行が終了すると，DOSに制御が戻ります．

<出力ファイル名>を指定する場合，その前に〝>″を1つまたは2つつけ加えます．このリダイレクト出力ファイルが指定されると，BASICの起動と共に前述の<プログラムファイル名>によって指定されたプログラムがロードされ，実行が開始されたときに次のような現象が生じます．

- プログラム内に，
  PRINT
  PRINT USING
  WRITE
  の各文がある場合，これらの文によって出力されるデータは，ディスプレイ画面に表示されると共に，<出力ファイル名>では指定されたリダイレクト出力ファイルも書き込まれます．

- <出力ファイル名>に〝SCRN：″を指定した場合は，データはディスプレイ画面に表示されるのみとなります．

- <出力ファイル名>の前に〝>>″を指定した場合，リダイレクト出力されるデータは，ファイルに追加書込み（アペンド）されます．

このように，入出力のリダイレクトを指定することにより，例えば定型化されたデータを入力するときなどにキー入力の代りにファイルから読み込ませたり，ディスプレイ画面に出力されるデータをファイルにとっておいたりすることが可能となります．

なお，リダイレクトによるプログラムの実行中に，STOPキーあるいはCTRL+Cを押すと，開かれているすべてのファイルは閉じられ，メッセージ〝Break in line xxxx″がディスプレイ画面に出力されて，BASICはDOSに戻ります．

また，エラーメッセージは，リダイレクト出力ファイルに書き込まれることはなく，常にディスプレイ画面にのみ出力されます．

例）BASIC MYPROG>DATA.OUT

INPUT文，LINE INPUT文などの入力文は，キーボードから入力されるデータを読みます．PRINT文によるデータ出力はファイルDATA.OUTに書き込まれます．

BASIC MYPROG<DATA.IN

INPUT文，LINE INPUT文などの入力文は，ファイルDATA.INからデータを読みます．PRINT文は，ディスプレイ画面にデータを出力します．

BASIC MYPROG<MYINPUT.DAT>MYOUTPUT.DAT

INPUT文，LINE INPUT文などの入力文は，ファイルMYINPUT.DATからデータを読み，PRINT文はファイルMYOUTPUT.DATにデータを書き込みます．

BASIC MYPROG<¥SALES¥JOHN¥TRANS>>¥SALES¥SALES.DAT

INPUT文，LINE INPUT文などの入力文は，ファイル¥SALES¥JOHN¥TRANSからデータを読みます．PRINT文はファイル¥SALES¥SALES.DATにデータを追加書き込み（アペンド）します．

第9章 機械語プログラムの使い方

N_{100}-BASIC には機械語プログラムを使うための機能として，POKE, BLOAD, BSAVE, CALL, DEF　SEG の 5 つの文と，PEEK，USR，VARPTR の 3 つの関数が用意されています．ここでは，機械語プログラムに使うメモリの確保の方法，機械語プログラムのメモリへの格納の方法，機械語プログラムの呼び出し方，および機械語プログラムと BASIC プログラムの間のデータの受渡しの方法について説明します．

なお，ここでの説明を正しく理解するには8086の機械語命令，およびメモリアドレッシングについての知識を必要とします．

9.1　メモリの確保

機械語プログラムを使うには，まずそのための領域をメモリ内に確保することが必要です．

まず最初に BASIC および DOS が，どのようにメモリを使っているかを説明しましょう．次のメモリマップを見てください．



この図の〝BASICワークスペース〟とは，BASICのプログラムとデータ（変数）を格納するために用いられる領域で，その大きさは最大64kバイトです．メモリの最上部には〝RAMディスク〟の領域があり，その大きさはDOSの立ち上げ時に決められます．RAMディスクの量の設定についてはMS-DOS ユーザーズガイドのSETUPコマンドを見てください．ワークスペースとRAMディスクの間の領域はBASICでは使われません．つまり，この〝未使用領域〟が機械語プログラムで使えるメモリになります．

未使用領域の大きさはDOSのCHKDSKコマンドで調べた使用可能なメモリの量からBASICのインタープリタの大きさ（約72000バイト）とBASICのワークスペース（デフォルトは65536バイト）をひいた値です．この値が機械語プログラムに必要なメモリ量以上あれば，この領域を機械語プログラム用として使用することができます．

未使用領域が十分にない場合は，BASICのワークスペースを小さくする必要があります．この目的のためには，BASICの起動時に／M：スイッチを使用するか，またはBASIC上でCLEAR文を使います．／M：スイッチで指定するメモリの上限，あるいはCLEAR文の第2パラメータは，BASICが使うワークスペースの量を決定します．たとえば，CHKDSKコマンドで調べた使用可能なメモリ量が120000バイトで，機械語プログラムに1024バイト必要なときには，ワークスペースの量を，120000－72000－1024＝46976バイトにします．すなわち，BASICを起動した後で，

```
CLEAR  , 46976
```

と指定するか，またはBASICの起動時に，

```
BASIC／M: 46976
```

と指定します．

さて，これで機械語プログラムに必要な領域が確保できました．次にその領域がメモリのどこにあるかを知る必要があります．メモリの上限番地はPC-100に実装されているメモリの量からわかります．368Kバイトのシステムならば5FFFFH番地です．RAMディスクの量が128Kバイト(20000H)ならば，RAMディスクの先頭番地は40000H番地です．この値から確保した機械語プログラム領域の大きさをひけばその領域の先頭番地が求まります．その大きさが1024バイト(400H)ならば3FC00H番地，すなわち，3FC0：0000番地が機械語領域の先頭番地になるわけです．

9.2 機械語プログラムの格納

機械語プログラムをメモリに格納するにはPOKE文，またはBLOAD文を使います．

POKE文を使うには，セグメントベースをDEF SEG文で指定しておく必要があります．POKE文で指定される番地は，そのセグメントベースの中からのオフセット値と解釈されます．

POKE文を使った機械語プログラムの例を示します．このプログラムは機械語プログラムを3FC0：0000番地から格納し実行します．

プログラムリスト

```
10 '
20 ' ＰＯＫＥ文による機械語プログラムの作成
30 '
40 '
50 ' セグメント番地の設定
60 '
70 ' 機械語プログラムは３ＦＣ０：００００番地から始まる
80 '
90 '
100 DEF SEG=&H3FC0
110 '
120 ' 機械語プログラムの書込み
130 '
140 FOR A=0 TO 34
150 READ A$:POKE A,VAL("&H"+A$)
160 NEXT A
170 CLS
180 LOCATE 10,1:PRINT "Look at the ";
190 '
200 ' 機械語プログラムの実行番地のオフセットの設定
210 '
220 DEF USR=0
230 '
240 ' 機械語プログラムの実行
250 '
260 A=USR(0)
270 LOCATE 10,20:PRINT "letters"
280 END
290 '
300 ' 機械語プログラム
310 '
320 ' "ＢＩＧ" という文字列を縦横を倍の大きさで書く
330 '
340 '
350 ' MOV CL,18H             ' 横幅を倍にするモードに入る
360 ' INT 0DCH               ' ソフトウェア割込みの２２０でＢＩＯＳを呼ぶ
370 '
380 DATA b1,18,cd,dc
390 '
400 ' MOV CL,1AH             ' 高さを倍にするモードに入る
410 ' INT 0DCH
420 '
430 DATA b1,1a,cd,dc
440 '
450 ' MOV AL,"B"             ' "Ｂ"を表示する
460 ' MOV CL,10H
470 ' INT 0DCH
480 '
490 DATA b0,42,b1,10,cd,dc
500 '
510 ' MOV AL,"C"             ' "Ｉ"を表示する
520 ' MOV CL,10H
530 ' INT 0DCH
540 '
550 DATA b0,49,b1,10,cd,dc
```

```
560 '
570 ' MOV AL,"G"                    ' "G"を表示する
580 ' MOV CL,10H
590 ' INT 0DCH
600 '
610 DATA b0,47,b1,10,cd,dc
620 '
630 ' MOV CL,19H                    ' 横幅を普通のサイズに戻す
640 ' INT 0DCH
650 '
660 DATA b1,19,cd,dc
670 '
680 ' MOV CL,1BH                    ' 高さを普通のサイズに戻す
690 ' INT 0DCH
700 '
710 DATA b1,1b,cd,dc
720 '
730 ' RETF                          ' 機械語プログラムは
740 '                                      必ず"FARRET"で終了
750 DATA cb
```

```
run
Look at the BIG letters
Ok
```

BLOAD文を使って機械語プログラムをメモリに格納するときには，あらかじめ，機械語プログラムファイルを作っておく必要があります．以下の例のように，まずPOKE文でメモリに機械語プログラムを書いておき，それをBSAVE文でファイルにします．一度機械語プログラムファイルを作れば，そのプログラムを使うプログラムでは単にそのファイルをBLOAD文で読み込むだけで，機械語プログラムをメモリに格納することができます．

プログラムリスト

```
10 '
20 ' ＢＳＡＶＥ文による機械語プログラムファイルの作成
30 '
40 '
50 ' セグメント番地の設定
60 '
70 ' 機械語プログラムは３ＦＣ０：００００番地に作られる
80 '
90 '
100 DEF SEG=&H3FC0
110 '
120 ' まず，機械語プログラムをＰＯＫＥ文で書き込む
130 '
140 FOR A=0 TO 34
150 READ A$:POKE A,VAL("&H"+A$)
160 NEXT A
200 '
210 ' メモリ上の機械語プログラムをファイルに書き込む
```

```
220 '
230 BSAVE "BIGCHAR.COD",0,35
260 END
290 '
300 ' 機械語プログラム
310 '
320 ' "ＢＩＧ" という文字列を縦横を倍の大きさで書く
330 '
340 '
350 ' MOV CL,18H                 ' 横幅を倍にするモードに入る
360 ' INT 0DCH                   ' ソフトウェア割込の２２０でＢＩＯＳを呼ぶ
370 '
380 DATA b1,18,cd,dc
390 '
400 ' MOV CL,1AH                 ' 高さを倍にするモードに入る
410 ' INT 0DCH
420 '
430 DATA b1,1a,cd,dc
440 '
450 ' MOV AL,"B"                 ' "Ｂ"を表示する
460 ' MOV CL,10H
470 ' INT 0DCH
480 '
490 DATA b0,42,b1,10,cd,dc
500 '
510 ' MOV AL,"C"                 ' "Ｉ"を表示する
520 ' MOV CL,10H
530 ' INT 0DCH
540 '
550 DATA b0,49,b1,10,cd,dc
560 '
570 ' MOV AL,"G"                 ' "Ｇ"を表示する
580 ' MOV CL,10H
590 ' INT 0DCH
600 '
610 DATA b0,47,b1,10,cd,dc
620 '
630 ' MOV CL,19H                ' 横幅を普通のサイズに戻す
640 ' INT 0DCH
650 '
660 DATA b1,19,cd,dc
670 '
680 ' MOV CL,1BH                ' 高さを普通のサイズに戻す
690 ' INT 0DCH
700 '
710 DATA b1,1b,cd,dc
720 '
730 ' RETF                      ' 機械語プログラムは必ず
740 '                                     必ず“ＦＡＲＲＥＴ”で終了
750 DATA cb
```

プログラムリスト

```
10 '
20 ' ＢＬＯＡＤ文による機械語プログラムの作成
30 '
40 '
50 ' セグメント番地の設定
60 '
70 ' 機械語プログラムは３ＦＣ０：００００番地に作られる
80 '
90 '
100 DEF SEG=&H3FC0.
110 '
120 ' 機械語プログラムファイルを読み込む
130 '
140 BLOAD "BIGCHAR.COD",0         ' 確実にＤＥＦ　ＳＥＧ文で
150                               ' 指定したセグメントに
160                               ' 読み込まれるように,
170                               ' ＯＦＦＳＥＴ　０を指定する
180 LOCATE 10,10:PRINT "Look at the ";
190 DEF USR=0
200 A=USR(0)
210 LOCATE 10,29:PRINT "letters"
220 END
```

BLOAD 文を使う場合，ファイルが実際にロードされる絶対アドレスは BLOAD 文の第 2 パラメータ（オフセット番地）の有無で変わります．上の例のように，オフセット番地を指定した場合には DEF SEG 文で指定したセグメントベースにそのオフセット番地を加えた値が実際のロード番地となります．オフセット番地を省略した場合は，そのファイルを BSAVE 文でセーブしたときのアドレスがそのまま使われます．すなわちその時点でのセグメントベースに BSAVE 文の第 2 パラメータで指定したオフセット番地を加えた値が実際のロード番地となります．

9.3 USR関数による機械語プログラムの呼び出し

機械語プログラムは USR 関数，または CALL 文で呼び出します．

機械語プログラムが USR 関数で呼び出されたときには AL レジスタに USR 関数の引数の型を示す情報がはいっています．AL の値と型の関係は次のようになっています．

AL の値	引数の型
2	整数
3	文字
4	単精度実数
8	倍精度実数

引数が数値のとき，すなわち AL が 2，4，8 のときには引数の値は FAC(浮動小数点アキュムレータ）と呼ばれる 8 バイトの領域に格納されており，このとき BX レジスタにこの FAC のオフセット番地，DS レジスタにそのセグメント値がはいっています．引数は FAC の中に以下に説明するように格納されています．

① 整数型のとき（AL＝2）

下位8ビット	上位8ビット

↑〔DS：BX＋2〕　↑〔DS：BX＋3〕

数値は16ビットの2の補数表現です.

② 単精度実数型のとき（AL＝4）

指数部	仮数の上位(第1)バイト	仮数の中位バイト	仮数の下位バイト

↑〔DS：BX〕　↑〔DS：BX＋3〕

数値は10進の浮動小数点形式で，指数部のMSB（最上位ビット）が仮数の符号，指数部の残り7ビットが示す数(2進表現)−64が実際の指数の値です．仮数部は24ビット長で，4ビットが1桁の数を表し，上位バイトのMSBの左に小数点があるとみなされます.

たとえば，引数が0.123456のとき，FACに渡される引数の表現は次のようになります.

40H	12H	34H	56H

↑〔DS：BX〕　↑〔DS：BX＋3〕

引数が−0.123456のときは引数は次のようになります.

C0H	12H	34H	56H

↑〔DS：BX〕　↑〔DS：BX＋3〕

③ 倍精度実数型のとき（AL＝8）

指数部	仮数の最上位(第1)バイト	≈	仮数の最下位(第7)バイト

↑〔DS：BX〕　↑〔DS：BX＋7〕

仮数部が56ビット長であることを除けば，単精度実数型と同じ形式です.

④ 文字型のとき（AL＝3）

引数が文字のときには，DXレジスタがストリングディスクリプタと呼ばれる3バイトの領域のオフセット番地，DSがそのセグメント値を持っています．ストリングディスクリプタの内容は次のようになっています.

〔DS：DX〕	文字列の長さ
〔DS：DX+1〕	文字列の置かれている領域のオフセット番地の下位8ビット
〔DS：DX+2〕	文字列の置かれている領域のオフセット番地の上位8ビット

なお，文字列そのものはストリングディスクリプタと同じセグメント内にあります．

USR関数で呼び出された機械語プログラムが値をBASICプログラムに返すときにはその型が引数の型と同じであることが必要です．返す値は引数と同じ形式でFACに格納します．返す値が文字列のときは渡されたディスクリプタが指す文字列の内容を変えることはできますが，長さを変えることはできませんので注意してください．

文字列を変更する場合には，そのUSR関数の引数に文字定数，または文字変数そのものを使ってはいけません．たとえば次の例は誤りです．

```
A$= "Bad"
B$=USR(A$)
```

必ず次のように2つ以上の項からなる「式」の形にするようにしてください．

```
A$= "Good"
B$=USR(A$+ "")
```

USR関数で呼ばれる機械語プログラムでは次の5つの注意を守ってください．

① プログラムは〝FAR PROCEDURE″にします．すなわち，プログラムからBASICに戻るときにはインターセグメントのRET命令を使う必要があります．
② すべてのセグメントレジスタの値を保存しなくてはなりません．機械語プログラムからBASICに戻るときには，すべてのセグメントレジスタの値がそのプログラムが呼ばれたときと同じ値になっていることが必要です．
③ SP（スタックポインタ）の値を保存する必要があります．
④ 機械語プログラムが割込みを禁止した場合，BASICに戻るときに再び割込み可能状態に戻さなければなりません．
⑤ BASICは16バイト分のスタックを機械語プログラム用に準備します．機械語プログラムがそれ以上にスタックを使う場合，機械語プログラムの領域に別のスタック領域を確保してください．このとき，BASICに戻る前にSPをもとの値に戻すことを忘れないでください．

なお，USR関数で機械語プログラムを呼び出したときには，DSレジスタは前述のとおりBASICのワークスペースのセグメント値を持っており，ES，SSもDSと同じ値に設定されています．CSはDEF SEG文で指定されたセグメント値になっています．

DEF SEG文を使わずに，USR関数を使った場合，CSはDS，ES，SSと同じくBASICのワークスペースのセグメント値になります．すなわち，DEF SEG文を用いずにUSR関数を使った場合，その機械語プログラムはBASICのワークスペースと同じセグメントにあるとみなされます．

プログラムリスト

```
10 '
20 ' ＵＳＲ関数のサンプルプログラム
30 '
40 ' ２つの文字列の長さを足し，Ｃ％に代入する
50 '
60 '
70 ' 機械語プログラムは３ＦＣ０：００００番地から
80 '                                   書き込まれる
90 '
100 DEF SEG=&H3FC0
110 '
120 ' 機械語プログラム（１）をメモリに書き込む
130 '
140 DEF USR1=0
150 FOR I=0 TO &HB
160 READ A$:POKE I,VAL("&h"+A$)
170 NEXT I
180 '
190 ' 機械語プログラム（２）をメモリに書き込む
200 '
210 DEF USR2=&H10
220 FOR I=&H10 TO &H1B
230 READ A$:POKE I,VAL("&h"+A$)
240 NEXT I
250 '
260 ' 機械語プログラム（３）をメモリに書き込む
270 '
280 DEF USR3=&H20
290 FOR I=&H20 TO &H34
300 READ A$:POKE I,VAL("&h"+A$)
310 NEXT I
320 '
330 ' 文字変数Ａ＄の長さを求める
340 '
350 A$="Hi,"
360 C$=USR1(A$)
370 '
380 ' 文字変数Ｂ＄の長さを求める
390 '
400 B$="How are you today?"
410 D$=USR2(B$)
420 '
430 ' ２つの文字変数の長さの合計を求め，Ａ％に代入する
440 '
450 A%=USR3(0)
460 PRINT"二つの文字列の長さの和は"
470 PRINT A%
480 END
```

```
490 '
500 ' 機械語プログラム（1）
510 ' ＵＳＲ関数の引数（文字）の長さを１００Ｈ番地にしまう
520 '
530 ' PUSH DS          ; ＢＡＳＩＣのワークスペースの
540 '                                    セグメントを退避する
550 DATA 1E
560 ' MOV BX,DX        ; ストリングディスクリプタの番地を
570 '                                             ＢＸレジスタに入れる
580 DATA 89,D3
590 ' MOV AL,[BX]      ; 文字列の長さをＡＬに得る
600 DATA 8A,7
610 ' PUSH CS       ; 機械語領域のセグメント番地をＤＳに入れる
620 DATA E
630 ' POP DS
640 DATA 1F
650 ' MOV [100H],1     ; ３ＦＣ０：１００Ｈ番地に
660 '                                       文字列の長さをしまう
670 DATA A2,0,1
680 ' POP DS           ; ＤＳをもとに戻す D
690 DATA 1F
700 ' RETF             ; ＢＡＳＩＣに戻る D
710 DATA CB
720 '
730 ' 機械語プログラム（2）
740 ' ＵＳＲ関数の引数（文字）の長さを１０１Ｈ番地にしまう
750 '
760 ' PUSH DS     ; ＢＡＳＩＣのワークスペースのセグメントを退避する
770 DATA 1E
780 ' MOV BX,DX        ; ストリングディスクリプタの番地を
790 '                                           ＢＸレジスタに入れる
800 DATA 89,D3
810 ' MOV AL,[BX]      ; 文字列の長さをＡＬに得る
820 DATA 8A,7
830 ' PUSH CS        ; 機械語領域のセグメント番地をＤＳに入れる
840 DATA E
850 ' POP DS
860 DATA 1F
870 ' MOV [101H],1     ; ３ＦＣ０：１０１Ｈ番地に
880 '                                         文字列の長さをしまう
890 DATA A2,1,1
900 ' POP DS           ; ＤＳをもとに戻す
910 DATA 1F
920 ' RETF             ; ＢＡＳＩＣに戻る
930 DATA CB
940 '
950 ' 機械語プログラム（3）
960 ' １００番地の値と１０１番地の値を足し，ＵＳＲ関数の値とする
970 ' PUSH DS        ; ＢＡＳＩＣのワークスペースの
980 '                                         セグメントを退避する
990 DATA 1e
1000 ' PUSH CS        ; ＤＳを機械語領域の先頭に設定する
1010 DATA e
1020 ' POP DS
1030 DATA 1f
1040 ' MOV AL,[100H]; 文字列１の長さをＡＸに入れる
1050 DATA a0,0,1
```

```
1060 ' MOV AH,0
1070 DATA b4,0
1080 ' MOV CL,[101H]; 文字列1の長さをCXに入れる
1090 DATA 8a,e,1,1
1100 ' MOV CH,0
1110 DATA b5,0
1120 ' ADD AX,CX     ; それらを加える
1130 DATA 1,c8
1140 ' POP DS       ; DSをBASICのワークスペースに戻す
1150 DATA 1f
1160 ' MOV [BX+2],AX      ; 答えをFACに入れる
1170 DATA 89,47,2
1180 ' RETF               ; BASICに戻る
1190 DATA CB
```

9.4　CALL文による機械語プログラムの呼び出し

機械語プログラムはCALL文を使って呼び出すこともできます．

CALL文は任意の数の引数を機械語プログラムに渡すことができます．引数は，その変数の置かれているオフセット番地(VARPTR関数で求めたときの番地と同じもの)が，スタックに積まれて，機械語プログラムに渡されます．CALL文がN個の引数を持っているとき，スタックの中は次のようになっています．

〔SS：SP〕	BASICへの戻り番地(4バイト)
〔SS：SP+4〕	引数Nのオフセット番地(2バイト)
〔SS：SP+6〕	引数N－1のオフセット番地(2バイト)
〔SS：SP+2*(N－1)+2〕	引数2のオフセット番地(2バイト)
〔SS：SP+2*(N－1)+4〕	引数1のオフセット番地(2バイト)

CALL文で呼び出された機械語プログラムが値をBASICに返すためには，オフセット番地で示された変数の内容を直接書き変えてやります．したがって，CALL文中の引数の並びの中には，機械語プログラムから返される値を格納するための変数を含めておかなくてはなりません．ただし，引数が文字変数の場合は，USR関数の場合と同じようにその変数の長さを変えることはできません．

機械語プログラムからBASICに戻るときには，〝FAR RET n〟命令を使います．ここでnはスタックに積まれた引数の並びの長さで，引数の数がN個のときにはn＝2＊Nになります．これはスタックから引数の並びを取り除くために必要です．

前節で述べたUSR関数を使うときの注意事項はすべてCALL文で呼ばれる機械語プログラムにも適用されます．さらに，CALL文を使うときには，CALL文が機械語プログラムに引数の数，型について何らの情報も渡さないことに特に注意してください．すなわち，CALL文で渡される引数の数と型は，機械語プログラムが期待する引数の数と型に完全に一致していることが必要です．

プログラムリスト

```
10 '
20 ' ＣＡＬＬ文のサンプルプログラム
30 '
40 ' ２つの文字列の長さを足し，Ａ％に代入する
50 '
60 '
70 ' 機械語プログラムは３ＦＣ０：００００番地から書き込まれる
80 '
90 DEF SEG=&H3FC0
100 '
110 ' 機械語プログラムをメモリに書き込む
120 '
130 FOR I=0 TO &H19
140 READ A$:POKE I,VAL("&h"+A$)
150 NEXT I
160 '
170 ' 文字変数Ａ＄，Ｂ＄を準備する
180 '
190 A$="Hi,"
200 B$="How are you today?"
210 '
220 ' ２つの文字変数の長さの合計を求め，Ａ％に代入する
230 '
240 CODE=0
250 CALL CODE (A%,A$,B$)
260 PRINT A%
270 END
280 '
290 ' 機械語プログラム（１）
300 ' ＵＳＲ関数の引数（文字）の長さを１００Ｈ番地にしまう
310 '
320 ' MOV BP,SP    ； ＢＰに引数リストの先頭番地をいれる
330 '              ； 引数リストは次のようになっている
340 '
350 '               ―――――――――――――――――――――――――
360 '  ＤＳ：ＳＰ   |戻り番地                                 |
370 '               |―――――――――――――――――――――――|
380 '  ＤＳ：ＳＰ＋４|Ｂ＄のストリングディスクリプタへの       |
390 '               |ポインタ                                 |
400 '               |―――――――――――――――――――――――|
410 '  ＤＳ：ＳＰ＋６|Ａ＄のストリングディスクリプタへの       |
420 '               |ポインタ                                 |
425 '               |―――――――――――――――――――――――|
430 '  ＤＳ：ＳＰ＋８|Ａ％へのポインタ                         |
440 '               ―――――――――――――――――――――――――
450 DATA 89,E5
460 ' MOV BX,[BP+4]      ； Ｂ＄のストリングディスクリプタの番地を
465 '                                                  ＢＸにいれる
470 DATA 8B,5E,4
480 ' MOV AL,[BX]        ； Ｂ＄の長さをＡＬにいれる
490 DATA 8A,7
500 ' MOV BX,[BP+6]      ； Ａ＄のストリングディスクリプタの番地を
505 '                                                  ＢＸにいれる
510 DATA 8B,5E,6
```

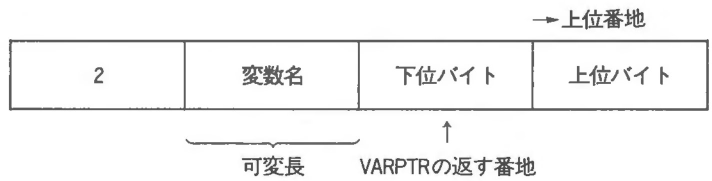
```
520 ' MOV CL,[BX]          ; A$の長さをCLにいれる
530 DATA 8A,F
540 ' MOV AH,0             ; AL，CLをワードに変換する
550 DATA B4,0
560 ' MOV CH,0
570 DATA B5,0
580 ' ADD AX,CX            ; A$とB$の長さを加える
590 DATA 1,C8
600 ' MOV BX,[BP+8]        ; A%の番地をBXにいれる
610 DATA 8B,5E,8
620 ' MOV [BX],AX          ; 答えをA%にいれる
630 DATA 89,7
640 ' RETF 6               ; 引数リストをスタックから取り除き，
645 '                                          BASICに戻る
650 DATA CA,6,0
```

9.5　VARPTR関数

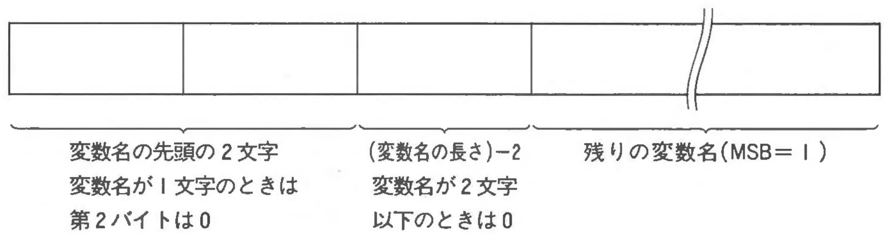
VARPTR 関数は指定された変数やファイルバッファの置かれているオフセット番地を返します．このときのセグメントは BASIC のワークスペースのセグメントと同じになります．

変数はメモリ中に次のように格納されています．

①　整数型変数



変数名は可変長で次のような形式で格納されています．



たとえば，“ABCD”，“G” という変数名はそれぞれ次のような形で格納されています．

41H	42H	2	C3H	C4H	47H	0	0

したがって変数A%が100のとき，A%のメモリ中の形式は次のようになっています．

2	41H	0	0	64H	0

↑ VARPTRの返す番号（64H）

② 単精度実数型変数

→上位番地

4	変数名	指数部	仮数の第1バイト	仮数の第2バイト	仮数の第3バイト

変数名：可変長　　指数部：↑ VARPTRの返す番地

指数，仮数の内部表現は次のとおりです．

◎仮数の符号＝指数部のMSB（0で正，1で負）

◎指数＝(指数部の下位7ビット)−64

◎仮数は第1バイトの左に小数点があると仮定します．

たとえば，変数B!の値が1.23456のときには，その変数はメモリ中で次のようになっています．

4	42H	0	0	41H	12H	34H	56H

↑ VARPTRの返す番地（41H）

③ 倍精度実数型変数

→上位番地

8	変数名	指数部	仮数の第1バイト	仮数の第2バイト	〜	仮数の第7バイト

変数名：可変長　　指数部：↑ VARPTRの返す番地

たとえば変数DOUBLE#の値が−1.23456789012340のときにはDOUBLE#はメモリ中で次のように格納されています．

8	44H	4FH	4	55H	42H	4CH	45H
	C1H	12H	34H	56H	78H	90H	12H	34H

↑ VARPTRの返す番地（C1H）

④　文字型変数

3	変数名	文字列の長さ	文字列の置かれているオフセット番地の下位8ビット	文字列の置かれているオフセット番地の上位8ビット

↑
VARPTRの返す番地

VARPTR関数の引数がファイル番号のときには，VARPTR関数はそのファイルのバッファのオフセット番地を返します．このバッファの最初の51バイトはBASICの作業用の領域となっており，ファイルとのデータ入出力に用いられる領域はVARPTR関数の値に51を加えた番地から始まります．

第10章
ファイルの階層構造

10.1　MS-DOSのファイルシステム

バージョン2.0以前のMS-DOSは，フロッピーディスクには十分な，単層のディレクトリ構造を持っています．しかし，メガバイト単位の大容量のハードディスク（5～20Mバイト）をサポートするには，このような少ないディレクトリ数では十分とはいえません．

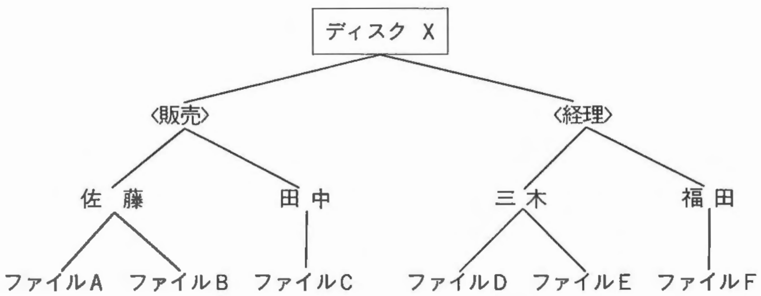
そこで，もっと良いファイル構造の必要性が出て来たわけです．

ただ単にディレクトリを大きくとるだけでは，あまり効果的な解決法とはいえません．例えば，1000ものファイルが1つのディレクトリ上にあるときに"DIR"を実行した場合のことを考えると，とてもユーザーフレンドリーであるとはいえません．

人間は，生来，組織図や家系図などのような階層的関係でものを考えます．ですから，ディスク上のファイルもこれと似たような方法で組み立てることができると良いでしょう．

例をあげて考えてみます．

あるビジネスで，販売部門と経理部門が大容量のディスク（名前を"X"とします）を備えた1台のコンピュータを共用し，各部門の部員が，販売報告の準備や，経理の情報整理のために，そのコンピュータを使うものとします．その時ディスクX上のファイルの構成を，次のような構造にするのが，自然でしょう．



バージョン2.0のMS-DOSではこのような構成をそのままファイルの構成として組み立てることができるように改良されました．そして，N_{100}-BASICでもそのままそのファイル構成を利用することができるのです．

10.1.1　ディレクトリとパス

N_{100}-BASICは，例えば仕事ごとの単位でディスク上のファイルを仕分けして格納しておくことができます．つまり，前項で見たように，全体のファイルをまず部員ごとの単位でまとめ，さらにその部員を部署ごとの単位でまとめるような構成をディスク上で実現できるのです．この場合，〝部員〟ごと，あるいは〝部署〟ごとにまとめるために，各部員の名前や各部署の名前を使うことになりますが，この名前を〝ディレクトリ〟と呼びます．前項の例で言えば，〝佐藤〟，〝三木〟，〝販売〟，〝経理〟などはみな，ディレクトリとなります．

ディレクトリの中には，ファイルはもちろん，さらに下位のディレクトリを置くこともでき，これらが多数重なって，階層構造を構成することになります．この場合，いちばん上位の根幹となるディレクトリのことを，〝ルートディレクトリ〟と呼びます．

さて，これでファイルを階層的に構成できることはわかりましたが，これらのファイルに対して読み書き（アクセス）するために必要なものが〝パス〟の概念です．すなわち，前項の例で，〝ディスクX上の，販売部の佐藤部員のレポートA〟にアクセスするためには，

ディスクX→販売部→佐藤→ファイルA

という順に見ていく必要があるわけですが，このファイルAにたどりつくまでの道筋を〝パス〟と呼び，このパスを正確に表現するものを〝パス名〟と呼びます．この場合のパス名は，

¥販売部¥佐藤¥

のように道筋を〝¥〟で区切って表現します．なお，いちばん初めの〝¥〟はルートディレクトリそのものを表すシンボルです．

ただこのようにいつもルートディレクトリからいちいちファイルを参照するような方法は，正確ではあっても煩雑です．そこで考えられたのが〝カレントディレクトリ〟と呼ばれるものです．すなわち，いったんあるディレクトリを選択したら，以後そこを中心にしてパスを指定するような方法です．例えば，前項の例で〝佐藤〟をカレントディレクトリに選んだとすると，ファイルAへのアクセスは，単に，

ファイルA

とすればよく，田中氏のファイルCへアクセスしたかったら，

..¥田中¥ファイルC

とすればよいのです．ここで〝..〟（ピリオド2個）は，〝ペアレントディレクトリ〟すなわちカレントディレクトリの一段階上位のディレクトリを表すシンボルです．これによれば，現在の状態（カレントディレクトリが〝佐藤〟）から経理部の福田氏のファイルFにアクセスするには，

..¥..¥経理部¥福田¥ファイルF

とすればよいことになります．

ただし，このように遠い場所にあるファイルを参照するには，かえってルートディレクトリからの指定方法の方がよいかもしれません．次の例，

¥経理部¥福田¥ファイルF

は上の例と同じファイルFをアクセスするものです．

ルートディレクトリから参照する場合は，カレントディレクトリがどこに設定されていてもかまいません．したがって，

ルートディレクトリから参照…………絶対指定
カレントディレクトリから参照………相対指定

ということが言えます．

なお，ルートディレクトリのペアレントディレクトリは，ルートディレクトリ自身です．

パスの深さ（ルートディレクトリから最も遠い位置にあるディレクトリまでのパスの長さ）に制限はありません．またユーザーの作成したディレクトリ内のファイル数についても制限はありません．

ただし，ディスク上の容量に制約がありますので，ディレクトリ数，ファイル数について，自ずと制限が生じます．

また，ルートディレクトリ中のファイル数（ディレクトリも含む）については，メディアごとに制限がありますので注意してください．

詳細については，MS-DOS プログラマーズリファレンスマニュアルをご覧ください．

10.1.2　ファイルスペック

N_{100}-BASIC でファイルを扱うコマンド，ステートメントにおいては，ファイルを指定するために〝ファイルスペック〟を用いますが，このファイルスペック中にパス名を含めることができます．すなわち，ファイルスペックの一般形は次のようになります．

〔<ドライブ名>：〕〔<パス名>〕<ファイル名>

〈パス名〉は前項で述べたように，ファイルの格納されているディレクトリを指し示す，¥で区切られた正しい文字列表現でなくてはなりません．

〈パス名〉は128バイトを超えてはなりません．もし128バイトを超えると，〝Bad file name〟エラーが生じます．

また，〈ドライブ名〉を指定するときは，必ず〈パス〉の前に置かなくてはいけません．次の例は間違っています．

¥SALES¥B：¥FILEA

このようなファイルスペックを指定すると，〝Bad file name〟エラーが生じます．

〈パス名〉が省略されるとカレントディレクトリ（現在選択されているディレクトリ．CHDIR コマンドで選択される）が仮定されます．

〈ドライブ名〉が省略されたときはカレントドライブが仮定されます．

10.1.3　ファイル操作のための命令

階層構造をもつディレクトリに格納されているディスクファイルを，パス名を含むファイルスペックで参照する命令には，次のようなものがあります．

FILES	ディレクトリ中のファイルをリストします．
KILL	ファイルを削除します．
NAME	ファイル名をつけ換えます．
LOAD	BASIC プログラムファイルをロードします．
SAVE	BASIC プログラムファイルをセーブします．
BLOAD	バイナリファイルをロードします．
BSAVE	バイナリファイルをセーブします．
CHAIN	別の BASIC プログラムに制御を渡します．
MERGE	ディスク上の BASIC プログラムファイルを，メモリ上のプログラムとマージします．
OPEN	シーケンシャルファイル，ランダムファイルを開きます．

これらの命令中，NAME 文，OPEN 文などはパス名の使用に制限があります．詳しくは N_{100}-BASIC リファレンスマニュアルの各文の項をご覧ください．

10.1.4　ディレクトリ操作のための命令

ディレクトリを階層的に作成したり，削除したり，カレントディレクトリを移動させたりするのに必要な命令として，次のようなものがあります．

MKDIR	新しいディレクトリを作成します．
RMDIR	ディレクトリを削除します．
CHDIR	カレントディレクトリを移動させます．

各命令のパラメータとしては，〈ドライブ名〉と参照すべきディレクトリへのパスを示す〈パス名〉を指定します．〈パス名〉は128バイトを超えることはできません．〈ドライブ名〉が省略されたときにはカレントドライブが仮定されます．

カレントディレクトリは，各ドライブごとに設定することができます．例えば，ドライブAについて，

```
CHDIR "A : ¥BIN¥DEMO"
```

を実行したのち，ドライブBについて，

```
CHDIR "B : ¥ASC¥TEST"
```

を実行したとします．現在のカレントドライブが〝A：″であるとして，パラメータなしの FILES 文，

```
FILES
```

を実行すると，〝A：¥BIN¥DEMO″ディレクトリ中のファイルがリストされ，

```
FILES "B :"
```

に対しては，〝B：¥ASC¥TEST″ディレクトリ中のファイルがリストされます．

また，いったんカレントディレクトリが設定されると，ドライブを移動してもそのドライブごとのカレントディレクトリが動くことはありません．先の例で，

SHELL "B :"

を実行し，カレントドライブをＢ：に移動してから，

FILES

を実行すると，"B：¥ASC¥TEST" ディレクトリ中のファイルがリストされ，

FILES "A :"

を実行すると，"A：¥BIN¥DEMO" ディレクトリ中のファイルがリストされることになります．

第11章
チャイルドプロセスのサポート

11.1　SHELL文

SHELL 文を使って，BASIC プログラムの実行中に別のプログラム (DOS 上で実行可能なプログラム) をロード，実行させることができます．別のプログラムの実行が終わると，コントロールは BASIC プログラムの SHELL 文の次の行にもどります．BASIC のコントロールの下に実行されたプログラムは，〝チャイルドプロセス〃として扱われます．

11.1.1　SHELL 文の書式

SHELL 文の書式は，以下のとおりです．

SHELL〔<文字列>〕

ここで<文字列>は走らせるプログラムのファイル名とオプション，パラメータ，引数を含んでいる正しい文字列表現です．

11.1.2　SHELL 文の動作

SHELL 文は，次のような規則にしたがって動作します．

(1)　<文字列>に入るプログラムファイル名には，拡張子をつけておいた方がよいでしょう．なぜならば，それがないとコマンドプロセッサが，COM ファイル，EXE ファイル，BAT ファイルの順に検索しなくてはならないからです．もし指定したファイルが見つからないと〝File not found error〃エラーが生じます．

(2)　プログラムのファイル名の後ろに1個以上のスペースがあけられておかれた文字列はみな，オプション，パラメータあるいは引数としてとり扱われます．

(3)　SHELL 文によってチャイルドプロセスが走っている間，BASIC はメモリ上に残っています．チャイルドプロセスが終了すると，再び BASIC に制御が戻ります．

(4)　〝BASIC. EXE〃，すなわち BASIC 自身に対して SHELL 文を実行することはできません．BASIC をチャイルドプロセスとして走らせようとすると，BASIC はロードの前にこのことをチェックし，次のメッセージを出した後，ペアレント（BASIC）にもどります．

〝You cannot run BASIC as child of BASIC〃

この制限は，ペアレントの BASIC を破壊しないようにするためのものです．

(5) 〈文字列〉の付いていないSHELL文を実行するとDOSのプロンプトが出力され，ユーザーはDOS上の操作であれば何でも行うことができます．BASICに戻るときはDOSの内部コマンド〝EXIT〟を使います．

例) SHELL⏎
A>DIR⏎
A>EXIT⏎
Ok

次に例としてDOSの〝SORT〟コマンドをSHELL文で起動してデータのソーティングを行ってみましょう．

```
100 OPEN "SORTIN.DAT" FOR OUTPUT AS#1
110 FOR I=1 TO 100
120 PRINT #1,RND(1)
130 NEXT I
140 CLOSE #1
150 SHELL "SORT<SORTIN.DAT>SORTOUT.DAT"
160 OPEN "SORTOUT.DAT" FOR INPUT AS#1
170 IF EOF(1) THEN END
180 INPUT#1,A
190 PRINT A
200 GOTO 170
```

11.1.3 エラー

SHELL文の実行時に起る可能性のあるエラーとしては，次のようなものがあります．

〝File not found〟

〈文字列〉で指定したプログラムのファイル名（.EXE, .COM）が見つかりませんでした．

〝Out of Memory〟

チャイルドを走らせるのに充分なメモリ容量がありません．

〝Can't continue after SHELL〟

BASICを続行するためのメモリ容量が不足しています．すべてのファイルが閉じられ，BASICはDOSに戻ります．これは，チャイルドプロセスが，実行終了後もメモリ上にとどまってしまっているときに起ります．

〝Internal Error〟

DOSが異常をおこしています．

11.1.4　チャイルドプロセスの使用上の注意

BASIC がチャイルドプロセスから自分自身を保護することは，完全にはできません．なぜなら，SHELL 文が実行されると，BASIC のコントロールする範囲外のところで，ファイルが開かれたり，各種のデバイスが使われたり，というようなじつに多くのことが行われるからです．以下にチャイルドプロセスが BASIC の実行環境を壊さないようにするための対策について述べます．

(1)　ハードウェア

一般に，SHELL 文実行中は，すべてのハードウェアは保護された状態にある事がのぞましいのですが，それには，〝SHLSAV〟と〝SHLRST〟というハードウェアに依存するローレベルのルーチンをチャイルドプロセス中で実行することにより，達成することができます．これらのルーチンは，SHELL 文の実行中，機械のステータス情報を，セーブ，レストアするようにするものです．また，ユーザーは，SHELL 文を使って実行されているチャイルドプロセス内で，特定のデバイスを使用しないようにする必要があります．

スクリーンデバイス（ディスプレイ）のチャイルドプロセスは，いろいろなスクリーンモードを変えてしまうことがあります．とくにスクリーンエディタが破壊されることがよくありますので注意してください．

他のハードウェアデバイス，たとえば割込みコントローラ，タイマー，DMA コントローラ，入出力スイッチ，UARTS などは BASIC によりある特定の状態におかれます．これらの機器は，ユーザーの知りうる範囲外においてチャイルドプロセスによって使用されることがあります．すなわち，ユーザーがこれらのデバイスを積極的に使用することをしなくても，チャイルドプロセスが自動的に使用してしまうことがあるのです．

(2)　ファイルシステム

BASIC（ペアレント）内でファイルを開けたままにしておくと，ファイルに操作を加えるようなチャイルドプロセスを実行したとき，たいへんな被害を被ることがあります．ファイルをチャイルドプロセスで操作する際には，SHELL 文を実行する前に必ずファイルを閉じ，SHELL 文の実行後に BASIC（ペアレント）にもどってきたら，再び開くようにしなくてはいけません．

(3)　メモリ管理

BASIC は SHELL 文により BASIC から別のコマンドへ移る前にメモリを開放しようとします．例外は／M：スイッチつきで BASIC が走らされた場合です．この場合 BASIC は，ユーザーが BASIC のメモリブロックの上限のさらに上の領域に何かロードするつもりである，ということを／M：スイッチによって認識するわけです．したがって，BASIC が SHELL 文の実行前にワークスペースを圧縮してしまう事態を防ぐことができます．このため，／M：スイッチを使うと SHELL 文は〝Out of memory〟エラーを起こすこともあります．

チャイルドプロセスは決して実行後メモリ上に残っていることがあってはいけません．もしそうであると，BASIC が，ワークスペースを元々のサイズに復帰させるための充分な場所がなくなってしまいます．BASIC がワークスペースをリストアできないと，ファイルはすべて閉じられ，〝SHELL can't continue〟というエラーメッセージが出力されて DOS に制御がもどります．

11.2 ENVIRON文

ENVIRON 文により，ユーザーは，BASICの環境文字列テーブルのパラメータを変えることができます．この文を使ってチャイルドプロセスの〝パス〞パラメータを変えたり，あるいは又，新しい環境パラメータを作ることにより，チャイルドプロセスに，パラメータを手渡したりすることができます．

11.2.1 ENVIRON 文の書式

ENVIRON 文の書式は次のとおりです．

ENVIRON〈パラメータ〉

ここで〈パラメータ〉とは新しい環境文字列パラメータを含む，正しい文字列表現です．

11.2.2 ENVIRON 文の動作

ENVIRON 文は，次のような規則にしたがって動作します．

(1) 〈パラメータ〉は〈パラメータ名〉=〈文字列〉の形の文字列でなくてはなりません．また，〈パラメータ〉に指定する文字列には，すべて大文字のキャラクタを使用しなければなりません．

(2) 〈パラメータ名〉は，例えば，〝PATH〞の様なパラメータの名前です．〈パラメータ名〉は，例えば，〝PATH=〞の様に〝=〞か〝　〞（スペース）によって〈文字列〉から分離されている必要があります．ENVIRON 文は，最初のスペースまたは〝=〞の左側にあるものはすべて〈パラメータ名〉とみなします．〈パラメータ名〉の後にくる最初の〝ノン・スペース〞または〝=でないもの〞が〈文字列〉とみなされます．

(3) 〈文字列〉は，新しいパラメータのテキストです．もし〈文字列〉が，空であったり，〝PATH=;〞の様に「;」（セミコロン）1個だけであったりすると，〈パラメータ〉で指定されるパラメータは，環境文字列テーブルから取り去られます．

(4) 〈パラメータ名〉で指定されるパラメータが存在しないと，〈パラメータ〉は，環境文字列テーブルの最後に付け加えられます．

(5) 〈パラメータ名〉で指定されるパラメータがもともとあったところに，同じパラメータ名を指定すると，環境文字列テーブルは圧縮され，新しい〈パラメータ〉がその終わりに付加されます．

(6) DOS の PATH コマンドあるいは SET コマンドで確保した環境文字列テーブルの大きさを越える文字列を ENVIRON 文でセットしようとすると，〝Out of memory〞エラーになります．

(7) BASIC の環境文字列テーブルは，BASIC の動作中に限り有効です．したがって，SHELL 文によって実行されるチャイルドプロセスに対してのみ，そのテーブルの内容が有効となります．

MS-DOS の内部コマンド〝PATH〞または〝SET〞によって変えられていない限り，BASIC 環境文字列テーブルは空です．まず DOS 上で以下の文を使って，ディスクA上のサブディレクトリに PATH を作ってみましょう．

```
PATH=A:¥SECOND¥THIRD
```

つぎにBASICを起動します.

ここで，次のようにするとパスが新しい値に変ります.

```
ENVIRON "PATH=A:¥SALES"
```

環境文字列テーブルは次のように変わります.

```
PATH=A:¥SALES
```

PATHパラメータはENVIRON文とENVIRON$関数をいっしょに使うと，アペンド（追加）することができます.

```
ENVIRON "PATH="+ENVIRON$("PATH")+ ";B:¥BLETCH"
```

環境文字列テーブルは次のようになります.

```
PATH=A:¥SALES;B:BLETCH
```

11.2.3　エラー

ENVIRON文の実行時に起る可能性のあるエラーとしては，次のようなものがあります.

"Type mismatch"

　〈パラメータ〉が文字列でないとき生じます.

"Out of memory"

　環境テーブルが，いっぱいで，これ以上割り付けできないとき生じます.

11.3　ENVIRON$関数

ENVIRON関数によりユーザーはBASICの環境テーブルから，指定された環境文字列を検索することができます.

11.3.1　ENVIRON$関数の書式

ENVIRON$関数の書式は次のとおりです.

```
ENVIRON$(<パラメータ名>)
```

または，

```
ENVIRON$(<パラメータ番号>)
```

〈パラメータ名〉は検索したいパラメータの名前を表す正しい文字式表現です.

〈パラメータ番号〉は1から255の範囲の整数を指定します.

11.3.2 ENVIRON$関数の動作

ENVIRON$関数は，次のような規則にしたがって動作します．

(1) 〈パラメータ名〉が使われると，ENVIRON$は環境文字列テーブルから，〈パラメータ名〉に続くテキストを含む文字列を返してきます．

(2) 〈パラメータ名〉＝が見つからないときや，〈パラメータ名〉＝の次にテキストがないときは，ヌルストリングを返します．

(3) 〈パラメータ番号〉が使われたときは，ENVIRON$は，文字列テーブルから〈パラメータ番号〉番目の環境文字列を返します．

DOS の内部コマンド〝PATH〞または〝SET〞によって変えられていなければ，BASIC の初期環境文字列は空です（ENVIRON 文の所を参照してください）．

ENVIRON 文の例を実行すると環境文字列テーブルは次のようになります．

```
PATH=A:¥SALES;A:¥ACCOUNTING;B:¥BLETCH
```

この場合次の文を実行すると，

```
PRINT ENVIRON$ ("PATH")
```

次の文字列が表示されます．

```
A:¥SALES;A:¥ACCOUNTING;B:¥BLETCH
```

そして次の文は，

```
PRINT ENVIRON$(1)
```

次の文字列を表示します．

```
PATH=A:¥SALES;A:¥ACCOUNTING;B¥BLETCH
```

以下に示すプログラムは，BASIC 環境文字テーブルを配列にセーブし，チャイルドプロセス用にテーブルを書き換えるようにしたものです．チャイルドプロセスの実行が終わると，さきにセーブした環境文字列をテーブルにリストア（再格納）します．

プログラムリスト

```
100 DIM ENV.TBLS(10)       '・・・・・１０　Ｐａｒｍをこえないように
110 N.PARMS=1              '・・・・・Ｐａｒｍの初期値
120 WHILE LEN(ENVIRON$(N.PARMS))>0
130 ENV.TBLS(N.PARMS)=ENVIRON$(N.PARMS)
140 N.PARMS=N.PARMS+1      '・・・・・
150 WEND
160 N.PARMS=N.PARMS-1
170 '＊＊＊ 新しいＥＮＶＩＲＯＮＭＥＮＴのストア ＊＊＊
180 ENVIRON "MYCHILD.PARM1=SORT BY NAME"
190 ENVIRON "MYCHILD.PARM2=LIST BY NAME"
                 ・
                 ・
                 ・
1000 SHELL "MYCHILD"       '・・・・・“ＭＹ　ＣＨＩＬＤ．ＥＸＥ”を走らせます
1010 FOR I=1 TO N.PARMS
1020 ENVIRON ENV.TBLS(I)   '・・・・・Ｐａｒｍのリストア
1030 NEXT I
                 ・
                 ・
                 ・
```

11.3.3　エラー

ENVIRON$関数の使用時に起る可能性のあるエラーとしては，次のようなものがあります．

"Illegal function call"

　＜パラメータ名＞が範囲外の値をとったとき生じます．

"Type mismatch"

　＜パラメータ名＞が文字列でないとき生じます．

"String too long"

　文字列が255バイトをこえたとき生じます．

付　録

コントロールコード表

キャラクタコード	対応するキー	機　　　能
1	CTRL + A	HELP キーと同じ.
2	CTRL + B	カーソルを１つ前のワードへ戻す.
3	CTRL + C	BASIC プログラム実行の中止. コマンド入力の中断. STOP キーと同じ.
5	CTRL + E	カーソルのある位置からその行の最後までを削除する.
6	CTRL + F	カーソルを次のワードに進める.
7	CTRL + G	スピーカを鳴らす.
8	CTRL + H	カーソルの左隣りの文字を消去する. BS キーと同じ.
9	CTRL + I	水平タブ. TAB キーと同じ. (途中に文字があっても消去しない.)
10	CTRL + J	ラインフィード.
11	CTRL + K	カーソルをホームポジションに移動する. HOME キーと同じ.
12	CTRL + L	画面クリア. CLR キーと同じ.
13	CTRL + M	キャリッジリターン. ↵ キーと同じ.
14	CTRL + N	カーソルを行の最後に移す.
18	CTRL + R	重ね書きモードとインサートモードの切り換え.
19	CTRL + S	表示を一時停止する（再開は任意のキーを入力する).
20	CTRL + T	ファンクションキーを表示する. KEY ON 文と同じ.
21	CTRL + U	１行キャンセル.
23	CTRL + W	カーソルのあるところから１ワード削除する.
24	CTRL + X	ロールアップ. SHIFT + ↑ キーと同じ.
25	CTRL + Y	ロールダウン. SHIFT + ↓ キーと同じ.
26	CTRL + Z	カーソルのある位置以降の画面に表示されている範囲を消去する.

NEC